C2

Wed Jan 02 17:30:26 2008 Function Index Page 2

Wed Jan 02 17:30:26 2008 Function Index Page 1

Wed Jan 02 17:30:26 2008 Function Index Page 2

Wed Jan 02 17:30:26 2008 File Index Page 2

Wed Jan 02 17:30:26 2008 File Index Page 1

Wed Jan 02 17:30:26 2008 File Index Page 2

```
/*****************************************************************************
**
** File Name:   EDMREProcMgrService.c
**
** Copyright (c) 1998,1999 by EMC Corporation.
**
** Purpose:
**      This module contains the Process Manager (thread) functions that
**      provide the top level processing of the 'asynchronous' Restore
**      Engine RPC's.  These functions are basically 'wrappers' for the
*       Restore Service Library calls that perform the actual RPC services.
**
** Table of Contents:
** ------------------
**              EDMRE_GetRestorableObjects
**              EDMRE_MarkObject
**              EDMRE_UnmarkObject
**              EDMRE_Submit
**              EDMRE_Start
**              EDMRE_Finish
**              EDMRE_FindRestorableObjects
**
**      Internal Functions:
**              EDMRE_ProgressCallback
**              EDMRE_RestoreCallback
**
** Compile-Time Options:
**
**
*****************************************************************************
*****/



/* The following provides an RCS id in the binary that can be located
** with the what(1) utility.  The intent is to keep this short.
*/

#ifndef lint
static char RCS_id [] = "$RCSfile$ "
                        "$Revision$ "
                        "$Date$" ;
#endif



/*
 * Feature test switches.
 * Standard defines required to turn on OS features go here.
 *
 * The following is required for code that uses POSIX API's.
 * Remove for non-POSIX, non-portable code.
 */

/* #define  _POSIX_SOURCE 1 */


/*
 * System headers.
 */
#include <stdlib.h>
#include <sys/syslog.h>
```



```
#include <unistd.h>
#include <string.h>

/*
 * Epoch headers.
 */

#include <ebutil/ebutil.h>
#include <restore/restore_engine.h>
#include <restore/REprogmsg.h>
#include <restore/EDMREProgressApi.h>

/*
 * Local headers
 */
#include <RSLapi.h>
#include <EDMProcessManager.h>
#include <EDMRestoreEngLog.h>
#include <EDMRECommandApi.h>


/*
 * Local functions
 */
static boolean_ty EDMRE_ProgressCallback( unsigned long progress );
static boolean_ty EDMRE_RestoreCallback( void );

/*****************************************************************************
**
** Routine:  EDMRE_GetAllBackupTimes
**
** Inputs:   void *input_args    ptr to struct with RPC input args
**
** Outputs:  void **status       addr of void * to receive ptr output arg struct
**
** Return Codes:
**           0 for success and non-zero for failure.
**
** Purpose:  Wrapper of Restore service library call to be executed
**            asynchronously from main RPC thread
**
*****************************************************************************
*/
int EDMRE_GetAllBackupTimes( void *input_args, void **output_args )
{
    RE_get_all_backup_times_args         *in_args
         = (RE_get_all_backup_times_args *)input_args;
    RE_get_all_backup_times_result       *out_args;

    int          status = COMMAND_RESULT_SUCCESS;

    out_args = calloc( 1, sizeof (RE_get_all_backup_times_result) );


    if (NULL == out_args)
    {
         EDMRestoreEng_logent(
                        __FILE__, __LINE__, LOG_ERR, MESSAGE_NO_MEMORY,
                0, "calloc fail for RE_get_all_backup_times_args" );
         status = COMMAND_RESULT_FAILURE;        /* fatal error */
    }

    else
```

```
123 2     {
124 2          out_args->cookie = in_args->cookie;
125 2          out_args->status = RSTSL_GetAllBackupTimes(
                                                    in_args->startTime,
126 2                                                 in_args->endTime,
127 2
                                                  in_args->maxEntries,
128 2                                                in_args->flags,
129 2
                                                &out_args->backupTimes,
130 2
                                                 &out_args->numEntries,
131 2
                                                    &out_args->cookie);
132 2          *output_args = (void *) out_args;
133 1     }

135 1     xdr_free( xdr_RE_get_all_backup_times_args, (char *)in_args);
136 1     free( in_args);

138 1     return status;
139     }
```



```
141   /*****************************************************************************
                                                                          ********
142   **
143   ** Routine:  EDMRE_GetRestorableObjects
144   **
145   ** Inputs:   void *input_args    ptr to struct with RPC input args
146   **
147   ** Outputs:  void **status       addr of void * to receive ptr output
                                                              arg struct
148   **
149   ** Return Codes:
150   **           0 for success and non-zero for failure.
151   **
152   ** Purpose:  Wrapper of Restore service library call to be executed
153   **           asynchronously from main RPC thread
154   **
155   *****************************************************************************
                                                                          ********
156   */
157   int EDMRE_GetRestorableObjects( void *input_args, void **output_args )
158 1 {
159 1     RE_get_restorable_objects_start_args          *in_args
160 1          = (RE_get_restorable_objects_start_args *)input_args;
161 1     RE_get_restorable_objects_output_result       *out_args;

163 1     int           status = COMMAND_RESULT_SUCCESS;

165 1     out_args = calloc( 1, sizeof(
                                RE_get_restorable_objects_output_result) );
166 1     if (NULL == out_args)
167 2     {
168 2          EDMRestoreEng_logent(
                          __FILE__, __LINE__, LOG_ERR, MESSAGE_NO_MEMORY,
169 2                    0,
               "calloc fail for RE_get_restorable_objects_output_result" );
170 2          status = COMMAND_RESULT_FAILURE;         /* fatal error */
171 1     }

173 1     else
174 2     {
175 2          out_args->cookie = in_args->cookie;
176 2          out_args->status = RSTSL_GetRestorableObjects(
177 2          (restorableObjectPtr)in_args->parentObj->RE_restorable_obj_u.
                                                                    tloInfo,
178 2                                          in_args->parentObj->objLevel,
179 2                                          &out_args->childrenObjs,
180 2                                          &out_args->cookie,
181 2                                          in_args->maxEntries,
182 2                                          &out_args->numEntries,
183 2                                          in_args->allowBadFiles );
184 2          *output_args = (void *)out_args;
185 1     }

187 1     xdr_free( xdr_RE_get_restorable_objects_start_args, (
                                                        char *)in_args);
188 1     free( in_args);

190 1     return status;
191   }
```

```
193  /*****************************************************************************
                                                                        ********
194  **
195  ** Routine:  EDMRE_MarkObject
196  **
197  ** Inputs:   void *input_args    ptr to struct with RPC input args
198  **
199  ** Outputs:  void **status       addr of void * to receive ptr output
                                                                    arg struct
200  **
201  ** Return Codes:
202  **           0 for success and non-zero for failure.
203  **
204  ** Purpose:  Wrapper of Restore service library call to be executed
205  **            asynchronously from main RPC thread
206  **
207  *****************************************************************************
                                                                        ********
208  */
209  int EDMRE_MarkObject( void *input_args, void **output_args )
210 1 {
211 1     RE_mark_object_args          *in_args = (
                                          RE_mark_object_args *)input_args;
212 1     RE_get_mark_results_result   *out_args;

214 1     int          status = COMMAND_RESULT_SUCCESS;

216 1     out_args = calloc( 1, sizeof(RE_get_mark_results_result) );
217 1     if (NULL == out_args)
218 2     {
219 2          EDMRestoreEng_logent(
                         __FILE__, __LINE__, LOG_ERR, MESSAGE_NO_MEMORY,
220 2                  0, "calloc fail for RE_get_mark_results_result" );
221 2          status = COMMAND_RESULT_FAILURE;        /* fatal error */
222 1     }

224 1     else
225 2     {
226 2          out_args->status = RSTSL_MarkObject( in_args->thisObj,
227 2                                               in_args->backupTime,
228 2                                               in_args->allowBadFiles,
229 2                                               in_args->descend,
230 2                                               &out_args->badFileCount,
231 2
                                            &out_args->permDenyFileCount,
232 2                                               &out_args->fileMarkCount,
233 2                                               &out_args->dirMarkCount,
234 2
                                              &out_args->otherMarkCount,
235 2                                               EDMRE_ProgressCallback );
236 2          *output_args = (void *)out_args;
237 1     }

239 1     xdr_free( xdr_RE_mark_object_args, (char *)in_args);
240 1     free( in_args );

242 1     return status;
243  }
```



```
245  /*****************************************************************************
                                                                        ********
246  **
247  ** Routine:  EDMRE_UnmarkObject
248  **
249  ** Inputs:   void *input_args    ptr to struct with RPC input args
250  **
251  ** Outputs:  void **status       addr of void * to receive ptr output
                                                                    arg struct
252  **
253  ** Return Codes:
254  **           0 for success and non-zero for failure.
255  **
256  ** Purpose:  Wrapper of Restore service library call to be executed
257  **            asynchronously from main RPC thread
258  **
259  *****************************************************************************
                                                                        ********
260  */
261  int EDMRE_UnmarkObject( void *input_args, void **output_args )
262 1 {
263 1     RE_unmark_object_args          *in_args = (
                                          RE_unmark_object_args *)input_args;
264 1     RE_get_unmark_results_result *out_args;

266 1     int          status = COMMAND_RESULT_SUCCESS;

268 1     out_args = calloc( 1, sizeof(RE_get_unmark_results_result) );
269 1     if (NULL == out_args)
270 2     {
271 2          EDMRestoreEng_logent(
                         __FILE__, __LINE__, LOG_ERR, MESSAGE_NO_MEMORY,
272 2                  0, "calloc fail for RE_get_unmark_results_result" );
273 2          status = COMMAND_RESULT_FAILURE;        /* fatal error */
274 1     }

276 1     else
277 2     {
278 2          out_args->status = RSTSL_UnmarkObject( in_args->thisObj,
279 2                                                 in_args->backupTime,
280 2                                                 in_args->badFilesOnly,
281 2                                                 in_args->descend,
282 2                                                 &out_args->badFileCount,
283 2                                                 &out_args->fileMarkCount,
284 2                                                 &out_args->dirMarkCount,
285 2
                                                &out_args->otherMarkCount,
286 2                                                 EDMRE_ProgressCallback );
287 2          *output_args = (void *)out_args;
288 1     }

290 1     xdr_free( xdr_RE_unmark_object_args, (char *)in_args);
291 1     free( in_args );

293 1     return status;
294  }
```

```
296  /*****************************************************************************
                                                                          ********
297  **
298  ** Routine: EDMRE_ProgressCallback
299  **
300  ** Inputs:   unsigned long progress      objects processed so far
301  **
302  ** Outputs:  none
303  **
304  ** Return Codes:
305  **           boolean_ty          FALSE if operation can continue
306  **                               TRUE if operation should be cancelled
307  **
308  ** Purpose:  Restore service library callback function to be called
                                                                    periodically
309  **            to return progress information and check for
                                                                  cancellation.
310  **
311  *****************************************************************************
                                                                          ********
312  */
313  static boolean_ty EDMRE_ProgressCallback( unsigned long progress )
314 1 {
315 1          UpdateProgressValue( progress );
316 1          return TestRpcCancelFlag( );
317  }
```



```
319  /*****************************************************************************
                                                                          ********
320  **
321  ** Routine: EDMRE_RestoreCallback
322  **
323  ** Inputs:   none
324  **
325  ** Outputs:  none
326  **
327  ** Return Codes:
328  **           boolean_ty          FALSE if operation can continue
329  **                               TRUE if operation should be cancelled
330  **
331  ** Purpose:  Restore service library callback function to be called
                                                                    periodically
332  **            by 'Start' function to return check for cancellation.
333  **
334  *****************************************************************************
                                                                          ********
335  */
336  static boolean_ty EDMRE_RestoreCallback( void )
337 1 {
338 1          EDMREGlobalStatus       internal_status;
339 1          long    last_rpc_time;
340 1          time_t  status_time;

342 1          internal_status = getGlobalStatus( &status_time );
343 1          last_rpc_time = getLastRpcTime( );

345 1          if (internal_status == EDMRE_STATE_USER_QUIT
346 1           || internal_status == EDMRE_STATE_ADMIN_QUIT)  /* someone
                                                                    aborted */
347 1                  return TRUE;
348 1          if (TRUE == IsRestoreTimedOut( last_rpc_time, status_time,
349 1                                         internal_status )
350 1          )
351 1                  return TRUE;
                                                    /* no sign of user life */
352 1          return TestRpcCancelFlag( );
                                                   /* user-signalled cancel */
353  }
```

```
355  /*****************************************************************************
                                                                          ********
356  **
357  ** Routine:  EDMRE_Submit
358  **
359  ** Inputs:   void *input_args    ptr to struct with RPC input args
360  **
361  ** Outputs:  void **status       addr of void * to receive ptr output
                                                                    arg struct
362  **
363  ** Return Codes:
364  **           0 for success and non-zero for failure.
365  **
366  ** Purpose:  Wrapper of Restore service library call to be executed
367  **            asynchronously from main RPC thread
368  **
369  *****************************************************************************
                                                                          ********
370  */
371  int EDMRE_Submit( void *input_args, void **output_args )
372 1 {
373 1     RE_submit_args               *in_args = (
                                                RE_submit_args *)input_args;
374 1     RE_get_submit_results_output         *out_args;
375 1     unsigned int                 object_count = 0;
376 1     EDMRST_submit_args           *submitArgs = calloc(1,sizeof(
                                                    EDMRST_submit_args));
377 1     int          status = COMMAND_RESULT_SUCCESS;

379 1     out_args = calloc( 1, sizeof(RE_get_submit_results_output) );
380 1     if (NULL == out_args)
381 2     {
382 2          EDMRestoreEng_logent(
                          __FILE__, __LINE__, LOG_ERR, MESSAGE_NO_MEMORY,
383 2                    0, "calloc fail for RE_get_submit_results_output" );
384 2          status = COMMAND_RESULT_FAILURE;        /* fatal error */
385 1     }

387 1     else
388 2     {
389 2         submitArgs->clientSocketPort = in_args->socketPort;
390 2          submitArgs->mapfile_env = esl_strdup(in_args->mapFile_env);
391 2          submitArgs->socketClientNm = esl_strdup(
                                             in_args->socketClientName);

393 2          out_args->submitObjectID = in_args->submitObjectID;
394 2          out_args->status = RSTSL_Submit( in_args->hostname,
395 2                                           in_args->overwritePolicy,
396 2                                           in_args->inPlace,
397 2                                           in_args->directory,
398 2                                           in_args->transport,
399 2                                           &out_args->submitObjectID,
400 2                                           &object_count,
401 2                                           EDMRE_ProgressCallback,
402 2                                           submitArgs);
403 2          out_args->objectsDone = object_count;
404 2          *output_args = (void *)out_args;
405 1     }

407 1     xdr_free( xdr_RE_submit_args, (char *)in_args);
408 1     free( in_args );

410 1     return status;
411  }
```



```
413  /*****************************************************************************
                                                                          ********
414  **
415  ** Routine:  EDMRE_Start
416  **
417  ** Inputs:   void *input_args    ptr to struct with RPC input args
418  **
419  ** Outputs:  void **status       addr of void * to receive ptr output
                                                                    arg struct
420  **
421  ** Return Codes:
422  **           0 for success and non-zero for failure.
423  **
424  ** Purpose:  Wrapper of Restore service library call to be executed
425  **            asynchronously from main RPC thread
426  **
427  *****************************************************************************
                                                                          ********
428  */
429  int EDMRE_Start( void *input_args, void **output_args )
430 1 {
431 1     RE_start_args        *in_args = (RE_start_args *)input_args;
432 1     RE_status_result     *out_args;

434 1     int          status = COMMAND_RESULT_SUCCESS;

436 1     out_args = calloc( 1, sizeof(RE_status_result) );
437 1     if (NULL == out_args)
438 2     {
439 2          EDMRestoreEng_logent(
                          __FILE__, __LINE__, LOG_ERR, MESSAGE_NO_MEMORY,
440 2                    0, "calloc fail for RE_status_result" );
441 2          status = COMMAND_RESULT_FAILURE;        /* fatal error */
442 1     }

444 1     else
445 2     {
446 2          out_args->status = RSTSL_Start( in_args->submitObjectID,
447 2                                          EDMRE_RestoreCallback );
448 2          *output_args = (void *)out_args;
449 1     }

451 1     xdr_free( xdr_RE_start_args, (char *)in_args);
452 1     free( in_args );

454 1     return status;
455  }
```

```
457  /*****************************************************************
                                                                ********
458  **
459  ** Routine:  EDMRE_Finish
460  **
461  ** Inputs:   void *input_args     OPTIONAL ptr to struct with RPC input
                                                                    args
462  **
463  ** Outputs:  void **status       OPTIONAL addr of void * to receive
                                                                  ptr to
464  **                               output arg struct
465  **
466  ** Return Codes:
467  **           0 for success and non-zero for failure.
468  **
469  ** Purpose:  Wrapper of Restore service library call to be executed
470  **            asynchronously from main RPC thread
471  **
472  *****************************************************************
                                                                ********
473  */
474  int EDMRE_Finish( void *input_args, void **output_args )
475 1 {
476 1         int     status = COMMAND_RESULT_SUCCESS;
477 1         RE_status_result        *out_args;

479 1         out_args = calloc( 1, sizeof(RE_status_result) );

481 1         if ( (out_args->status = RSTSL_Finish( )) != E_SUCCESS)
482 1                 status  = COMMAND_RESULT_FAILURE;

484 1         if (NULL != input_args)
485 2         {
486 2                 xdr_free( xdr_RE_null_args, (char *)input_args);
487 2                 free( input_args );
488 2                 *output_args = (void *)out_args;
489 1         }
490 1         else    /* only keep output struct if user want it */
491 1                 free( out_args );

493 1         return status;
494  }
```



```
496  /*****************************************************************
                                                                ********
497  **
498  ** Routine:  EDMRE_FindRestorableObjects
499  **
500  ** Inputs:   void *input_args    ptr to struct with RPC input args
501  **
502  ** Outputs:  void **status       addr of void * to receive ptr output
                                                              arg struct
503  **
504  ** Return Codes:
505  **           0 for success and non-zero for failure.
506  **
507  ** Purpose:  Wrapper of Restore service library call to be executed
508  **            asynchronously from main RPC thread
509  **
510  *****************************************************************
                                                                ********
511  */
512  int EDMRE_FindRestorableObjects(
                                    void *input_args, void **output_args )
513 1 {
514 1         RE_find_restorable_objects_args         *in_args =
515 1                         (RE_find_restorable_objects_args *)input_args;
516 1         RE_find_restorable_objects_result       *out_args;
517 1         EBREC_SearchCriteriaRec
                                                          searchCriteria;

519 1         int     status = COMMAND_RESULT_SUCCESS;

521 1         out_args = calloc( 1, sizeof(
                                    RE_find_restorable_objects_result) );
522 1         if (NULL == out_args)
523 2         {
524 2                 EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
525 2                                       MESSAGE_NO_MEMORY, 0,
526 2                         "calloc fail for
                                    RE_find_restorable_objects_result" );
527 2                 status = COMMAND_RESULT_FAILURE;
                                                        /* fatal error */
528 1         }

530 1         else
531 2         {
532 2                 /* prepare search criteria structure for input to
                                                          RSTST func: */
533 2                 strncpy( searchCriteria.startDirectory,
534 2                         in_args->searchCriteria->startDirectory,
                                                                256);
535 2                 searchCriteria.descendDirectory =
536 2
                            in_args->searchCriteria->descendDirectory;
537 2                 strncpy( searchCriteria.searchString,
538 2                         in_args->searchCriteria->searchString, 128);
539 2                 searchCriteria.excludeString =
540 2
                          in_args->searchCriteria->excludeString;
541 2                 searchCriteria.typeOfFile =
542 2                           in_args->searchCriteria->typeOfFile;
543 2                 strncpy( searchCriteria.owner,
544 2                         in_args->searchCriteria->owner, 64);
545 2                 searchCriteria.excludeOwner =
546 2                           in_args->searchCriteria->excludeOwner;
547 2                 strncpy( searchCriteria.group,
548 2                         in_args->searchCriteria->group, 64);
```

```
549 2                   searchCriteria.excludeGroup =
550 2                                   in_args->searchCriteria->excludeGroup;
551 2                   searchCriteria.sizeInBytes.high =
552 2                                   in_args->searchCriteria->sizeInBytes.
                                                                        high;
553 2                   searchCriteria.sizeInBytes.low =
554 2                                   in_args->searchCriteria->sizeInBytes.
                                                                        low;
555 2                   searchCriteria.sizeMatch =
556 2                                   in_args->searchCriteria->sizeMatch;
557 2                   searchCriteria.startTime =
558 2                                   in_args->searchCriteria->startTime;
559 2                   searchCriteria.endTime =
560 2                                   in_args->searchCriteria->endTime;

562 2                   out_args->status = RSTSL_FindRestorableObjects(
563 2
                                                        &searchCriteria,
564 2
                                                    EDMRE_ProgressCallback );
565 2                   *output_args = (void *)out_args;
566 1           }

568 1           xdr_free( xdr_RE_find_restorable_objects_args, (
                                                        char *)in_args );
569 1           free( in_args );

571 1           return status;
572       }
```



```
576   /*****************************************************************************
                                                                          ********
577   **
578   ** Routine:  int EDMRE_Load_recx_directives
579   **
580   ** Inputs:  RE_recx_file_info *fileinfo   Information on file to be
                                                                        retreived
581   **
582   ** Outputs: Error or success from the RSTSL call
583   **
584   ** Return Codes:
585   **           0 for success and non-zero for failure.
586   **
587   ** Purpose:  Function to retrieve directives file from client and then
588   **           load the file contents into the context structure.
                                                                        The file
589   **           transfer is done with edm link.
590   **
591   *****************************************************************************
                                                                          ********
592   */
593   int EDMRE_Load_recx_directives( void *input_args,
594                                   void **output_args )
595 1 {
596 1   RSTRPC_recx_file_info * fileinfo = (
                                        RSTRPC_recx_file_info *)input_args;
597 1   RE_status_result *outargs;
598 1   outargs = calloc(1,sizeof(RE_status_result));

600 1   /*
601 1    * Actually load the recx structure.
602 1    */
603 1   outargs->status = RSTSL_Load_recx_directives(fileinfo);
604 1   *output_args = (void *)outargs;

606 1   /*
607 1    * Return that the RPC was atleast successfule,
                                                    the load may not have been
608 1    */
609 1   return COMMAND_RESULT_SUCCESS;

611   }
```

```
614  /*****************************************************************************
                                                                          ********
615  **
616  ** Routine:  EDMRE_SetPreviousBackup
617  **
618  ** Inputs:   void *input_args    ptr to struct with RPC input args
619  **
620  ** Outputs:  void **status       addr of void * to receive ptr output
                                                                        arg struct
621  **
622  ** Return Codes:
623  **           0 for success and non-zero for failure.
624  **
625  ** Purpose:  Wrapper of Restore service library call to be executed
626  **            asynchronously from main RPC thread
627  **
628  *****************************************************************************
                                                                          ********
629  */
630  int EDMRE_SetPreviousBackup( void *input_args, void **output_args )
631 1 {
632 1      RE_set_backup_time_args      *in_args
633 1           = (RE_set_backup_time_args *)input_args;
634 1      RE_get_all_backup_times_result       *out_args;

636 1      int          status = COMMAND_RESULT_SUCCESS;

638 1      out_args = calloc( 1, sizeof (RE_status_result) );


641 1      if (NULL == out_args)
642 2      {
643 2           EDMRestoreEng_logent(
                          __FILE__, __LINE__, LOG_ERR, MESSAGE_NO_MEMORY,
644 2                  0, "calloc fail for RE_status_result" );
645 2           status = COMMAND_RESULT_FAILURE;        /* fatal error */
646 1      }

648 1      else
649 2      {
650 2           out_args->status = RSTSL_SetPrevBackup( in_args->flags);
651 2           *output_args = (void *) out_args;
652 1      }

654 1      xdr_free( xdr_RE_set_backup_time_args, (char *)in_args);
655 1      free( in_args);

657 1      return status;
658  }
```



```
662  /*****************************************************************************
                                                                          ********
663  **
664  ** Routine:  EDMRE_SetBackupForTime
665  **
666  ** Inputs:   void *input_args    ptr to struct with RPC input args
667  **
668  ** Outputs:  void **status       addr of void * to receive ptr output
                                                                        arg struct
669  **
670  ** Return Codes:
671  **           0 for success and non-zero for failure.
672  **
673  ** Purpose:  Wrapper of Restore service library call to be executed
674  **            asynchronously from main RPC thread
675  **
676  *****************************************************************************
                                                                          ********
677  */
678  int EDMRE_SetBackupForTime( void *input_args, void **output_args )
679 1 {
680 1      RE_backup_for_time_args      *in_args
681 1           = (RE_backup_for_time_args *)input_args;
682 1      RE_get_all_backup_times_result       *out_args;

684 1      int          status = COMMAND_RESULT_SUCCESS;

686 1      out_args = calloc( 1, sizeof (RE_status_result) );


689 1      if (NULL == out_args)
690 2      {
691 2           EDMRestoreEng_logent(
                          __FILE__, __LINE__, LOG_ERR, MESSAGE_NO_MEMORY,
692 2                  0, "calloc fail for RE_status_result" );
693 2           status = COMMAND_RESULT_FAILURE;        /* fatal error */
694 1      }
695 1      else
696 2      {
697 2           out_args->status = RSTSL_SetBackupForTime( in_args->time,
698 2                                                      in_args->flags);
699 2           *output_args = (void *) out_args;
700 1      }

702 1      xdr_free( xdr_RE_backup_for_time_args, (char *)in_args);
703 1      free( in_args);

705 1      return status;
706  }
```

```
708  /*****************************************************************************
                                                                          ********
709  **
710  ** Routine:  EDMRE_SetNextBackup
711  **
712  ** Inputs:   void *input_args    ptr to struct with RPC input args
713  **
714  ** Outputs:  void **status       addr of void * to receive ptr output
                                                                  arg struct
715  **
716  ** Return Codes:
717  **           0 for success and non-zero for failure.
718  **
719  ** Purpose:  Wrapper of Restore service library call to be executed
720  **            asynchronously from main RPC thread
721  **
722  *****************************************************************************
                                                                          ********
723  */
724  int EDMRE_SetNextBackup( void *input_args, void **output_args )
725 1 {
726 1     RE_set_backup_time_args      *in_args
727 1          = (RE_set_backup_time_args *)input_args;
728 1     RE_get_all_backup_times_result       *out_args;

730 1     int           status = COMMAND_RESULT_SUCCESS;

732 1     out_args = calloc( 1, sizeof (RE_status_result) );


735 1     if (NULL == out_args)
736 2     {
737 2          EDMRestoreEng_logent(
                          __FILE__, __LINE__, LOG_ERR, MESSAGE_NO_MEMORY,
738 2                    0, "calloc fail for RE_status_result" );
739 2          status = COMMAND_RESULT_FAILURE;        /* fatal error */
740 1     }

742 1     else
743 2     {
744 2          out_args->status = RSTSL_SetNextBackup( in_args->flags);
745 2          *output_args = (void *) out_args;
746 1     }

748 1     xdr_free( xdr_RE_set_backup_time_args, (char *)in_args);
749 1     free( in_args);

751 1     return status;
752  }
```



```
754  /*****************************************************************************
                                                                          ********
755  **
756  ** Routine:  EDMRE_SetFirstBackup
757  **
758  ** Inputs:   void *input_args    ptr to struct with RPC input args
759  **
760  ** Outputs:  void **status       addr of void * to receive ptr output
                                                                  arg struct
761  **
762  ** Return Codes:
763  **           0 for success and non-zero for failure.
764  **
765  ** Purpose:  Wrapper of Restore service library call to be executed
766  **            asynchronously from main RPC thread
767  **
768  *****************************************************************************
                                                                          ********
769  */
770  int EDMRE_SetFirstBackup( void *input_args, void **output_args )
771 1 {
772 1     RE_set_backup_time_args      *in_args
773 1          = (RE_set_backup_time_args *)input_args;
774 1     RE_get_all_backup_times_result       *out_args;

776 1     int           status = COMMAND_RESULT_SUCCESS;

778 1     out_args = calloc( 1, sizeof (RE_status_result) );


781 1     if (NULL == out_args)
782 2     {
783 2          EDMRestoreEng_logent(
                          __FILE__, __LINE__, LOG_ERR, MESSAGE_NO_MEMORY,
784 2                    0, "calloc fail for RE_status_result" );
785 2          status = COMMAND_RESULT_FAILURE;        /* fatal error */
786 1     }

788 1     else
789 2     {
790 2          out_args->status = RSTSL_SetFirstBackup( in_args->flags);
791 2          *output_args = (void *) out_args;
792 1     }

794 1     xdr_free( xdr_RE_set_backup_time_args, (char *)in_args);
795 1     free( in_args);

797 1     return status;
798  }
```

```
800  /*****************************************************************
                                                                ********
801  **
802  ** Routine:  EDMRE_SetMostRecentBackup
803  **
804  ** Inputs:   void *input_args    ptr to struct with RPC input args
805  **
806  ** Outputs:  void **status       addr of void * to receive ptr output
                                                          arg struct
807  **
808  ** Return Codes:
809  **           0 for success and non-zero for failure.
810  **
811  ** Purpose:  Wrapper of Restore service library call to be executed
812  **            asynchronously from main RPC thread
813  **
814  *****************************************************************
                                                                ********
815  */
816  int EDMRE_SetMostRecentBackup( void *input_args, void **output_args )
817 1 {
818 1      RE_set_backup_time_args      *in_args
819 1           = (RE_set_backup_time_args *)input_args;
820 1      RE_get_all_backup_times_result       *out_args;

822 1      int          status = COMMAND_RESULT_SUCCESS;

824 1      out_args = calloc( 1, sizeof (RE_status_result) );


827 1      if (NULL == out_args)
828 2      {
829 2           EDMRestoreEng_logent(
                        __FILE__, __LINE__, LOG_ERR, MESSAGE_NO_MEMORY,
830 2                  0, "calloc fail for RE_status_result" );
831 2           status = COMMAND_RESULT_FAILURE;        /* fatal error */
832 1      }

834 1      else
835 2      {
836 2           out_args->status = RSTSL_SetMostRecentBackup( in_args->flags);
837 2           *output_args = (void *) out_args;
838 1      }

840 1      xdr_free( xdr_RE_set_backup_time_args, (char *)in_args);
841 1      free( in_args);

843 1      return status;
844  }
```

```
/*****************************************************************************
                                                                          ******
**
** File Name:   RSLsubmit.c
**
** Copyright (c) 1998,1999 by EMC Corporation.
**
** Purpose:
** -----------
**      The intent of the contents of this file is to implement the
**      functions to create the submitObject and submitElement.
**
**      These functions are provided to allow:
**      - creation of submit objects,
                                  which define the set of objects to be
**        restored and the scripts to be run before and after
                                                          restoration,
**
** The following functions comprise restoral management:
**
**      RSTSL_Submit
**
**
** Compile-Time Options:
**               This section must list any compile time definitions
**               which will affect this header.
**
*****************************************************************************
                                                                          *****/



/*
 * Feature test switches.
 * Standard defines required to turn on OS features go here.
 *
 * The following is required for code that uses POSIX API's.
 * Remove for non-POSIX, non-portable code.
 */

#define  _POSIX_SOURCE 1
#define ULONG_TO_CHAR_SIZE 16


/*
 * System headers.
 */
/* for CreateSubmitname */
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
/* for CreateSubmitname */
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/wait.h>

/*
 * Epoch headers.
 */
#include <eb/eb_port.h>
#include <eb/rb_log.h>
#include <ebutil/eb_normalize.h>
#include <ebutil/ebutil.h>
#include <ebreport/ebvl.h>
#include <restore/RSLplugin.h>
```



```
/*
 * Local headers
 */

#include <RSLinterns.h>
#include <restore/EDMRESubmitApi.h>




void
fill_client_dirtop(struct recover_context *rcx);

static int
push_submit_file(struct recover_context *rcx,
                 int fd,
                 struct mark_summary *this_submit_files,
                 ebvl_volidlist_ty **this_submit_volumes,
                 struct mark_summary *total_submit_files,
                 ebvl_volidlist_ty **total_submit_volumes,
                 RSTSL_SubmitProgressProc progressCB,
                 boolean_ty *submitCancelled);


static int
push_bfinfo_to_submitfile(struct recover_context *rcx,
                          int plane,
                          cat_descriptor *catd,
                          long lmno,
                          int fd,
                          ebfs_uid_ty *prev_ebd,
                          struct mark_summary *this_submit_files,
                          ebvl_volidlist_ty **this_submit_volumes,
                          struct mark_summary *total_submit_files,
                          ebvl_volidlist_ty **total_submit_volumes);

static int
push_to_submitfile(int fd,
                   char *buf,
                   uint_t nbytes);

static void
ebfsid2str_lz(ebfs_uid_ty *ebfsidp,
              register char *buf);

static int
ssID2ebfd(rbssID_t *ssidp,
          ebfs_uid_ty *ebfdp);




/*****************************************************************************
 * Restoral Management Functions:
 *
 *  These functions are provided to allow:
 *      - creation of submit objects,
                                  which define the set of objects to be
 *        restored and the scripts to be run before and after
                                                          restoration,
 *      - starting the restoral of a submit object.
 *
 * The following functions comprise restoral management:
 *
 *      RSTSL_Submit
 *      RSTSL_Start
```

```
127     *
128     ************************************************************************
129     * Submit
130     *
131     * This function creates a submit object from the currently marked
132     * restorable objects. It is passed to RSTSL_Start to begin execution
133     * of the restore.
134     *
135     * Parameters:
136     *
137     * policy        (I) - The overwrite policy to use
138     * inPlace       (
                  I) - Flag if the restoral is to be in original locations
139     * hostName      (I) - host to restore to (only if inPlace == False)
140     * directory     (I) - directory to restore to (
                                              only if inPlace == False)
141     * transport      (
              I) - Indicator of transport the restoral is to be over (SCSI
142     *                    or network)
143     * submitObjIDptr(
                    IO) - ID of the submit user object created to describe
144     *                    the restore
145     * ObjectsSubmitted (O) - number of total file objects submitted.
146     * progressCB    (
                  I) - pointer to callback function to report progress and
147     *                    test for cancellation
148     *
149     ************************************************************************/
150     eerrno_ty RSTSL_Submit(const char              *hostName,
151                            const OverwritePolicy   policy,
152                            const boolean_ty        inPlace,
153                            const char              *directory,
154                            const RestoreTransport  transport,
155                            int                      *submitObjID,
156                            unsigned int             *ObjectsSubmitted,
157                            RSTSL_SubmitProgressProc progressCB,
158                            EDMRST_submit_args       *submitArgs)
159   1 {

161   1    int submitElemID;
162   1    int ret_status;
163   1    int SEstatus = E_SUCCESS;
164   1    int SOstatus = E_SUCCESS;
165   1    int submit_fd;
166   1    char submit_filename[2048];
167   1    struct mark_summary *this_submit_files;
168   1    ebvl_volidlist_ty *this_submit_volumes = NULL;
169   1    struct mark_summary *total_submit_files;
170   1    ebvl_volidlist_ty *total_submit_volumes = NULL;
171   1    char *hostname_SE = NULL;
172   1    char wi_type;
173   1    boolean_ty submitCancelled = FALSE;
174   1    char **envVar;
175   1    int tmp;

177   1    /*
178   1     * The submitElement destructor should free these structures.
179   1     */
180   1    this_submit_files = malloc(sizeof(struct mark_summary));
181   1    total_submit_files = malloc(sizeof(struct mark_summary));

183   1    if((NULL == this_submit_files) ||
184   1       (NULL ==total_submit_files))
185   2    {
186   2      rec_api_log_csm(SUB_CSM_NOMEM, NULL);
187   2      rbe_log_stats(0, "Could not allocate memory RSTSL_Submit()");
```



```
188   2      return(EP_RB_RECOVER_NOMEM);
189   1    }

191   1    memset(this_submit_files, 0, sizeof(struct mark_summary));
192   1    memset(total_submit_files, 0, sizeof(struct mark_summary));



196   1    /*
197   1     * for Direct Connect, call its plugin version of Submit:
198   1     * Did not change plug in call for CLI arguments for Port and Name
199   1     */
200   1    if (rcp->rc_backup_app != 0)
201   1      return rcp-> currentPIptr-> piFuncArray[ PIFuncIndexSubmit ]
202   1                              ( rcp,
203   1                                hostName,
204   1                                policy,
205   1                                inPlace,
206   1                                directory,
207   1                                transport,
208   1                                submitObjID,
209   1                                progressCB );

211   1    *submitObjID = NewSubmitObject(&SOstatus);

213   1    if(E_SUCCESS != SOstatus)
214   2      {
215   2        rbe_log_stats(0, "Could not submit new object ID");
216   2        return (EP_RB_RECOVER_FATALERR);
217   1      }


220   1    if (!inPlace && ((hostName == NULL) || ('\0' == hostName[0])))
221   2      {
222   2        rbe_log_stats(0, "Hostname or inplace not set");
223   2        return(EP_RB_RECOVER_BAD_ARGS);
224   1      }

226   1    submitElemID = NewSubmitElement(*submitObjID, &SEstatus);

228   1    if(E_SUCCESS != SEstatus)
229   2      {
230   2        rbe_log_stats(0, "Could not create new submit elements");
231   2        return (EP_RB_RECOVER_FATALERR);
232   1      }


235   1    if(0 != SetSOBasics(*submitObjID,
236   1                        0,
237   1                        0,
238   1                        &SOstatus))

240   2      {
241   2        rbe_log_stats(0, "Could not set SO basics");
242   2        return (EP_RB_RECOVER_FATALERR);
243   1      }

245   1    /*
246   1     * Set up the environment variables
247   1     */
248   1    envVar = calloc(2,sizeof(char *));
249   1    envVar[0] = submitArgs->mapfile_env;
250   1    if (NULL != envVar[0])
251   2      {
252   2        if (0 == strcmp(envVar[0], "\0"))
253   3          {
```

```
254  3            envVar[0] = NULL;
255  2          }
256  1      }
257  1    envVar[1] = NULL;
258  1    if (NULL != envVar[0])
259  2      {
260  2        if (0 != SetSOExecutePhase(*submitObjID,NULL,NULL,envVar,&tmp))
261  3          {
262  3            rec_api_log_csm(FATAL_ERROR, NULL);
263  3            rbe_log_stats(
                       0, "Could not set the mapfile environment variable\n");
264  3            return(FATAL_ERROR);

266  2          }
267  1      }

269  1    /*
270  1     * Set the name of the client who initiated the call and the
271  1     * port to connect to.  Needs to be done before Direct connect call
272  1     */
273  1    if((0 != submitArgs->clientSocketPort) || (
                                        NULL != submitArgs->socketClientNm))
274  2      {
275  2        if ( 0 != SetSEEbciConnect(*submitObjID,
276  2                                   submitElemID,
277  2                                   submitArgs->socketClientNm,
278  2                                   submitArgs->clientSocketPort,
279  2                                   &SEstatus))
280  3          {
281  3            rbe_log_stats(
                           0, "Could not set socket port or client name");
282  3            return (EP_RB_RECOVER_FATALERR);
283  2          }
284  1      }

286  1      if(0 != SetSOAdminID(*submitObjID,
287  1                           (
           rcp->rc_recovery_flags & RC_RECFLAG_ADMINISTRATOR) ? 1 : 0,
288  1                           (
           rcp->rc_recovery_flags & RC_RECFLAG_SOURCE_SYSADMIN) ? 1 : 0,
289  1                           (
             rcp->rc_recovery_flags & RC_RECFLAG_DEST_SYSADMIN) ? 1 : 0,
290  1                          &SOstatus))
291  2      {
292  2        return (EP_RB_RECOVER_FATALERR);
293  1      }

295  1      if ((NULL != rcp->rc_human_uidname) &&
296  1          (NULL != rcp->rc_effective_uidname))
297  2      {
298  2        if(0 != SetSOUserID(*submitObjID,
299  2                            rcp->rc_human_uid,
300  2                            rcp->rc_human_uidname,
301  2                            rcp->rc_effective_uid,
302  2                            rcp->rc_effective_uidname,
303  2                            &SOstatus))
304  3          {
305  3            return (EP_RB_RECOVER_FATALERR);
306  2          }

308  1      }
309  1      else
310  2      {
311  2        rbe_log_stats(
          0, "Restore context has not set human_name and/or effective_name,
                                                          in RSTSL_Submit()");
```



```
312  2        return(EP_RB_RECOVER_BAD_CONTEXT);
313  1      }

315  1      if(0 != SetSOVMCheck(*submitObjID,
316  1                           (
                        rcp->rc_recovery_flags & RC_RECFLAG_NO_VM_CHECK),
317  1                           &SOstatus))

319  2      {
320  2        return (EP_RB_RECOVER_FATALERR);
321  1      }

323  1      /*
324  1       * Initialize this at the start of each recover within a single
325  1       * recovery session.
                              This is important if the destination directory
326  1       * gets changed from a specific location to simply "in place".
327  1       */

329  1      if (NULL != rcp->rc_client_dirtop)
330  2      {
331  2          free(rcp->rc_client_dirtop);
332  1      }

334  1      rcp->rc_client_dirtop = NULL;

336  1      if (rcp->rc_client_hostname)
337  2      {
338  2          free(rcp->rc_client_hostname);
339  1      }

341  2      {
342  2        /*
343  2         * We need to get information about this work item from
344  2         * from the config structure. Currently this is just the
345  2         * work item type.
346  2         */
347  2        RBC_WORKGROUP *wgp;
348  2        boolean_ty wi_found = FALSE;

350  2        if(NULL == rcp->rc_config)
351  3        {
352  3            rbe_log_stats(
        0, "Restore context has not set up the config is RSTSL_Submit()");
353  3            return(EP_RB_RECOVER_BAD_CONTEXT);
354  2        }

356  2        if(NULL == rcp->rc_workitem_name)
357  3        {
358  3            rbe_log_stats(
    0, "Restore context has not set up current work item in RSTSL_Submit(
                                                                        )");
359  3            return(EP_RB_RECOVER_BAD_CONTEXT);
360  2        }

362  2        wi_found = GetClientTypeFromConfig(rcp->rc_config,
363  2                                           rcp->rc_workitem_name,
364  2                                           &wi_type);
365  2        if (FALSE == wi_found)
366  3        {
367  3            rbe_log_stats(
                     0, "Could not find the work item in the config structure in
                                                                RSTSL_Submit()");
368  3            return(EP_RB_RECOVER_BAD_CONTEXT);
369  2        }
370  1      }
```

```
371 1     if (inPlace)
372 2     {
373 2       if(NULL == rcp->rc_source_client_hostname)
374 3       {
375 3          return(EP_RB_RECOVER_BAD_CONTEXT);
376 2       }
377 2       else
378 3       {
379 3          hostname_SE = esl_strdup(rcp->rc_source_client_hostname);
380 2       }

382 1     }
383 1     else /* !inPlace */
384 2     {
385 2       /* hostName is checked above if !inPlace */

387 2       hostname_SE = (char *) hostName;
388 1     }
389 1     if (NULL == (rcp->rc_client_hostname = esl_strdup((
                                                    char *)hostname_SE)))
390 2     {
391 2          rec_api_log_csm(SUB_CSM_NOMEM, NULL);
392 2          rbe_log_stats(0, "Could not allocate memory RSTSL_Submit()");
393 2          return(EP_RB_RECOVER_NOMEM);
394 1     }

396 1     rcp->rc_overwrite_policy = policy;

398 1     /*
399 1      * fill_client_dirtop must always be called because
400 1      * cross recoveries for NOS clients require some special
401 1      * handling. The destination server name needs to be
402 1      * prepended along with a : in order for it to build the
403 1      * correct target string on the recover command sent to the client.
404 1      * fill_client_dirtop() handles this correctly.
405 1      */

407 1     if(0 != fill_client_dirtop2(rcp->rc_config,
408 1                                 inPlace,
409 1                                 (!inPlace) ? (char *) directory : (
                                                      char *) NULL,
410 1                                 rcp->rc_workitem_name,
411 1                                 hostname_SE,
412 1                                 &(rcp->rc_client_dirtop)))
413 2     {
414 2          rbe_log_stats(0, "Internal error:"\
415 2                        "Could not file client dirtop in RSTSL_Submit(
                                                                  )");

417 2          return(EP_RB_RECOVER_BAD_CONTEXT);
418 1     }

420 1     if (NULL == rcp->rc_client_dirtop)
421 2     {
422 2          rec_api_log_csm(SUB_CSM_NOMEM, NULL);
423 2          rbe_log_stats(0, "Could not allocate memory RSTSL_Submit()");
424 2          return(EP_RB_RECOVER_NOMEM);
425 1     }

427 1     if((NULL != rcp->rc_workitem_name) &&
428 1        (NULL != rcp->rc_template_name) &&
429 1        (NULL != rcp->rc_source_client_hostname))
430 2     {
431 2       if ( 0 != SetSEBasics(*submitObjID,
432 2                             submitElemID,
433 2                             rcp->rc_workitem_name,
```



```
434 2                             rcp->rc_template_name,
435 2                             rcp->rc_saveset_thread,
436 2                             rcp->rc_source_client_hostname,
437 2                             wi_type,
438 2                             &SEstatus))

440 3       {
441 3          return (EP_RB_RECOVER_FATALERR);
442 2       }
443 1     }
444 1     else
445 2     {
446 2       rbe_log_stats(
                          0, "Restore context has not set template_name,
              witem_name and/or source_client_name, in RSTSL_Submit()");
447 2       return(EP_RB_RECOVER_BAD_CONTEXT);
448 1     }



452 1     /* directory & hostName are check as input args */
453 1     /* What about transport ?? */

455 1     if(0 != SetSEDestination(*submitObjID,
456 1                              submitElemID,
457 1                              (char *)hostname_SE,
458 1                              inPlace,
459 1                              (char *)directory,
460 1                              policy,
461 1                              transport,
462 1                              &SEstatus))
463 2     {
464 2       return (EP_RB_RECOVER_FATALERR);
465 1     }

467 1     if(NULL != rcp->rc_client_dirtop)
468 2     {
469 2       if(0 != SetSEDirtop(*submitObjID,
470 2                           submitElemID,
471 2                           rcp->rc_client_dirtop,
472 2                           &SEstatus))
473 3       {
474 3          return (EP_RB_RECOVER_FATALERR);
475 2       }
476 1     }
477 1     else
478 2     {
479 2       rbe_log_stats(
              0, "Restore context has not set dirtop, in RSTSL_Submit()");
480 2       return(EP_RB_RECOVER_BAD_CONTEXT);
481 1     }

483 1     if((NULL != rcp->rc_client_scriptname) &&
484 1        (NULL != rcp->rc_client_rbuname))
485 2     {
486 2       if(0 != SetSEScriptName(*submitObjID,
487 2                               submitElemID,
488 2                               rcp->rc_client_scriptname,
489 2                               rcp->rc_client_rbuname,
490 2                               &SEstatus))
491 3       {
492 3          return (EP_RB_RECOVER_FATALERR);
493 2       }
494 1     }

496 1     else
```

```
497 2     {
498 2       rbe_log_stats(
             0, "Restore context has not set scriptname or client user_name,
                                                          in RSTSL_Submit()");
499 2       return(EP_RB_RECOVER_BAD_CONTEXT);
500 1     }

502 1     /*
503 1      * Progress callback intended to test for cancelation and
504 1      * to report progress on the submit to the user.
505 1      */
506 1     if(TRUE == progressCB(0))
507 2     {
508 2       /* Lets clean up here!
509 2        * right now there is no clean up routine for submitObjects.
510 2        */
511 2       rbe_log_stats(EP_RB_RECOVER_ABORT,"User abort during submit.");
512 2       submitCancelled = TRUE;
513 2       return(EP_RB_RECOVER_ABORT);
514 1     }
515 1     submit_fd = OpenSubmitFile(TRUE,
516 1                                *submitObjID,
517 1                                submitElemID,
518 1                                &SEstatus);


521 1     if(-1 == submit_fd)
522 2     {
523 2       return (EP_RB_RECOVER_FATALERR);
524 1     }

526 1     ret_status = push_submit_file(rcp,
527 1                                   submit_fd,
528 1                                   this_submit_files,
529 1                                   &this_submit_volumes,
530 1                                   total_submit_files,
531 1                                   &total_submit_volumes,
532 1                                   progressCB,
533 1                                   &submitCancelled);


536 1     CloseSubmitFile(submit_fd, TRUE, &SEstatus);

538 1     if(TRUE == submitCancelled)
539 2     {
540 2       /* Lets clean up here!
541 2        * right now there is no clean up routine for submitObjects.
542 2        */
543 2       rbe_log_stats(EP_RB_RECOVER_ABORT,"User abort during submit.");
544 2       return(EP_RB_RECOVER_ABORT);
545 1     }

547 1     if(-1 == ret_status)
548 2     {
549 2       return (EP_RB_RECOVER_FATALERR);
550 1     }

552 1     *ObjectsSubmitted = (unsigned int) ret_status;

554 1     if(0 != SetSOTotalSize(*submitObjID,
555 1                            total_submit_files,
556 1                            &SOstatus))
557 2     {
558 2       ; /* Not sure this one needs to be handled */
559 1     }
```



```
561 1     if(0 !=SetSOTotalVolumes(*submitObjID,
562 1                              total_submit_volumes,
563 1                              &SOstatus))
564 2     {
565 2       ; /* Not sure this one needs to be handled */
566 1     }

568 1     if(0 != SetSESummary(*submitObjID, submitElemID,
569 1                          this_submit_files, &SEstatus))
570 2     {
571 2       ; /* Not sure this one needs to be handled */
572 1     }
573 1     if(0 != SetSEVolumes(*submitObjID,
574 1                          submitElemID,
575 1                          this_submit_volumes,
576 1                          &SEstatus))
577 2     {
578 2       ; /* Not sure this one needs to be handled */
579 1     }

581 1     return( E_SUCCESS );
582   }
```

```
586   void
587   fill_client_dirtop(struct recover_context *rcx)
588 1 {
589 1     char buf[4096];
590 1     RBC_WORKITEM *pwi;
591 1     RBC_WORKGROUP *pwg;
592 1     boolean_ty cross_recover;


595 1     for (pwg = rcx->rc_config->pgrouplist; NULL != pwg;
                                                        pwg = pwg->next)
596 2     {
597 2          for (pwi = pwg->pwilist; NULL != pwi; pwi = pwi->next)
598 3          {
599 3              if (0 == strcmp(pwi->name, rcx->rc_workitem_name))
600 4              {
601 4                  goto gotit2;
602 3              }
603 2          }
604 1     }

606 1 gotit2:

608 1     /*
609 1      * if the dirtop already specifies a netware client
610 1      * target, remove it first.
611 1      */

613 1     if (rcx->rc_client_dirtop != NULL
614 1          && NULL != strchr(rcx->rc_client_dirtop, ':'))
615 2     {
616 2          return;
617 1     }

619 1     /*
620 1      * cross_recover is a boolean variable used to indicate a
                                                    cross-recover request.
621 1      * This will set the proper target for NOS clients and should NOT
                                                        affect others.
622 1      */

624 1     cross_recover = (NULL != rcx->rc_client_hostname) &&
625 1                 (0 != strcmp(
              rcx->rc_source_client_hostname, rcx->rc_client_hostname));

627 1     sprintf(buf, "%s%s%s",
628 1            ((NULL != pwi && NULL != pwi->nw_clnt_target)
629 1                ? (cross_recover
630 1                    ? rcx->rc_client_hostname
631 1                    : pwi->nw_clnt_target)
632 1                : ""),
633 1              ((NULL != pwi && NULL != pwi->nw_clnt_target) ? ":" : ""),
634 1              ((
        NULL != rcx->rc_client_dirtop) ? rcx->rc_client_dirtop : "/"));

636 1     if (rcx->rc_client_dirtop != NULL)
637 2     {
638 2          free (rcx->rc_client_dirtop);
639 1     }

641 1     rcx->rc_client_dirtop = esl_strdup(buf);
642 1     if (NULL == rcx->rc_client_dirtop)
643 2     {
```



```
644 2         rec_api_log_csm(SUB_CSM_NOMEM, NULL);
645 1     }
646   }         /* fill_client_dirtop */
```

```
651  static int
652  push_submit_file(struct recover_context *rcx,
653                   int submit_fd,
654                   struct mark_summary *this_submit_files,
655                   ebvl_volidlist_ty **this_submit_volumes,
656                   struct mark_summary *total_submit_files,
657                   ebvl_volidlist_ty **total_submit_volumes,
658                   RSTSL_SubmitProgressProc progressCB,
659                   boolean_ty *submitCancelled)
660  1 {

662  1   int submit_cont = 1;
663  1   int retStatus = 0;
664  1   int bitfiles_pushed = 0;
665  1   ebfs_uid_ty prev_ebd;

667  1   if(NULL != submitCancelled)
668  1     *submitCancelled = FALSE;
669  1   else
670  1     return -1;

672  1   memset(&prev_ebd, 0, sizeof(ebfs_uid_ty));


675  1   if((NULL == rcx) ||
676  1      (NULL == this_submit_files) ||
677  1      (NULL == this_submit_volumes) ||
678  1      (NULL == total_submit_files) ||
679  1      (NULL == total_submit_volumes))
680  2   {
681  2     return -1;
682  1   }

684  1   if (submit_cont)
685  2   {

687  2          int plane;

689  2         /* MCAT_TOP is #define in mcat.h */
690  2          for (plane = (-rcx->rc_nplanes)+1; plane <= MCAT_TOP;
                                                                  plane++)
691  3          {
692  3              cat_descriptor *catd = mcat_getcatd(rcx->rc_mcp, plane);
693  3              char *marks = rcx->rc_marks[-plane];
694  3              long ntrlm;
695  3              long lmno;


698  3              if (catd == NULL)    /* should never happen */
699  4              {
700  4                  continue;
701  3              }
702  3              /* catdesc.h */
703  3              ntrlm = catd_ntrlm(catd);

705  3              for (lmno = 0; lmno < ntrlm; lmno++)
706  4              {
707  4                /* RSLrbrmain.h */
708  4                  if (! TLMNO_MARKED(marks, lmno))
709  5                  {
710  5                      continue;
711  4                  }
```



```
713  4                  retStatus = push_bfinfo_to_submitfile(rcx,
714  4                                                        plane, catd,
                                                                  lmno,
715  4                                                        submit_fd,
716  4                                                        &prev_ebd,
717  4                                                      this_submit_files,
718  4                                                    this_submit_volumes,
719  4                                                     total_submit_files,
720  4                                                   total_submit_volumes);
721  4                  if (1 == retStatus)
722  5                  {
723  5                    bitfiles_pushed++;
724  4                  }
725  4                  if(bitfiles_pushed % 1024)
726  5                  {
727  5                    if(TRUE == progressCB(bitfiles_pushed))
728  6                    {

730  6                     rbe_log_stats(EP_RB_RECOVER_ABORT,
731  6                                   "User abort during submit.");
732  6                      *submitCancelled = TRUE;
733  6                      return(-1);
734  5                    }

736  4                  }
737  3              }
738  2          }

740  2          if(-1 == retStatus)
741  3          {
742  3            return -1;
743  2          }
744  2          return bitfiles_pushed;
745  1      }
746     }
```

```
749  /*
750   * Returns: -1 for file not submitted for restore, error encountered.
751   *           0 for file not submitted for restore,
                                                    NO error encountered.
752   *           1 for file subitted successfully.
753   */
754  static int
755  push_bfinfo_to_submitfile(struct recover_context *rcx,
756                            int plane,
757                            cat_descriptor *catd,
758                            long lmno,
759                            int fd,
760                            ebfs_uid_ty *prev_ebd,
761                            struct mark_summary *this_submit_files,
762                            ebvl_volidlist_ty **this_submit_volumes,
763                            struct mark_summary *total_submit_files,
764                            ebvl_volidlist_ty **total_submit_volumes)
765  1 {
766  1   /* Add mark support */
767  1     char              ebfsbfstr[34];
768  1     rbtree_elem_t     tlm;
769  1     rbcat_elem_t      clm;
770  1     long              catlmno;
771  1     cat_descriptor  *catlm_catd;
772  1     char              ebfsdirstr[34];
773  1     char              buf[100];
774  1     size_t            nbytes;
775  1     size_t            namesize = 0;
776  1     char             *fname = "<name unknown>";
777  1     char              *this_file;
778  1     eperrno           ep_status;
779  1     ebfs_uid_ty ebd;
780  1     ebfs_uid_ty zero_bitfileid;
781  1     char              *directives_list=NULL;
782  1     int               directives_size=0;


785  1     (void)catd_read_trlm(catd, lmno, &tlm, &this_file, (size_t *)NULL);

787  1     /*
788  1      * Get the corresponding catalog element.
789  1      * Note that it might come from a different
790  1      * plane if this tree element is a DS_NONE.
791  1      */

793  1     if (tlm.te_catelem != -1)
794  2     {
795  2          /*
796  2           * not DS_NONE -- the common case
797  2           */

799  2          catlmno = tlm.te_catelem;
800  2          catlm_catd = catd;
801  1     }
802  1     else
803  2     {
804  2          int clmplane;

806  2          /*
807  2           * This is a DS_NONE record.  Get
808  2           * the real corresponding catalog element.
809  2           */

811  2          dsnone_get_realcat(rcx, lmno, plane, &catlmno, &clmplane);
```



```
813  2          /*
814  2           * There is a special case for the root ("/")
815  2           * in the backup catalogs.  It's in the tree file
816  2           * but not in any catalog file.  This case can also
817  2           * occur for leading directories that are "above"
818  2           * the starting point for a work-item.  Silently
819  2           * ignore such directories, unless debugmode is on.
820  2           */

822  2          if (catlmno == -1)
823  3          {
824  3              size_t len;

826  3              (void)catd_read_trlm(catd, lmno, &tlm, &fname, &len);
827  3              if (len != 0 && debugmode) /* len 0 filters out "/" */
828  4              {
829  4                /*rbe_log_stats(
                   0, "** warning: cannot find catalog record for file %s;"
830  4                          " skipping it.", fname);*/
831  3              }
832  3              return 0;
833  2          }
834  2          catlm_catd = mcat_getcatd(rcx->rc_mcp, clmplane);
835  1      }
836  1      (void)catd_read_catlm(catlm_catd, catlmno, &clm, (char **)NULL);

838  1      add_to_summary(this_submit_files, &clm);
839  1      add_to_summary(total_submit_files, &clm);

841  1      *this_submit_volumes = ebvl_genvolidlist(*this_submit_volumes,
842  1                                               &clm.ce_bitfileID,
843  1                                               1,
                                                    ebvl_EbfsIdType_BitFile,
844  1                                               &ep_status);

846  1      if((NULL == *this_submit_volumes) || (0 != ep_status))
847  2      {
848  2        rbe_log_stats(
                  0, "** warning: cannot maintain volume list for submit."
849  2                     " skipping it.");
850  1      }

852  1      *total_submit_volumes = ebvl_genvolidlist(*total_submit_volumes,
853  1                                                &clm.ce_bitfileID,
854  1                                                1,
                                                     ebvl_EbfsIdType_BitFile,
855  1                                                &ep_status);

857  1      if((NULL == *total_submit_volumes) || (0 != ep_status))
858  2      {
859  2        rbe_log_stats(
                  0, "** warning: cannot maintain volume list for submit."
860  2                     " skipping it.");
861  1      }


864  1      memset(&zero_bitfileid, 0, sizeof(ebfs_uid_ty));


867  1      if(0 == memcmp(&zero_bitfileid,
868  1                     &clm.ce_bitfileID,
869  1                     sizeof(ebfs_uid_ty)))
870  2      {
871  2        (void)catd_read_catlm(catlm_catd, catlmno, &clm, &fname);
```

```
873  2        rbe_log_stats(
                     0, "** warning: There is no backup data to recover for "
874  2                    "file \"%s\"; skipping it.", fname);
875  2        return 0;
876  1      }


879  1      /*
880  1       * If this is a renamed element,
                                        must get the current name from cat
881  1       */


884  1      (void)catd_read_catlm(catlm_catd, catlmno, &clm, &fname);
885  1      if (clm.ce_status & CESTAT_RENAME)
886  2        {
887  2           /*
888  2            * name may contains substrings as in netware
889  2            */
890  2           namesize = (size_t)clm.ce_namelen;
891  1        }

893  1      if(0 != ssID2ebfd(&clm.ce_ssID, &ebd))
894  2      {
895  2        rbe_log_stats(
               0, "** warning: could not determine bitfile directory for "
896  2                    "file \"%s\"; skipping it.", fname);
897  2        return 0;
898  1      }

900  1      if ((NULL != rcx->recx_directives_p) || (NULL != fname))
901  2        {
902  2           directives_list=RSTSL_get_directives_for_file(
                                                  rcx->recx_directives_p,
903  2                                                  fname);
904  1        }
905  1      else
906  2        {
907  2           directives_list = NULL;
908  1        }
909  1      /* done in case string not null terminated */
910  1      if(directives_list !=NULL)
911  2        {
912  2           directives_size = strlen(directives_list);
913  1        }
914  1      else
915  2        {
916  2           directives_size = 0;
917  1        }

919  1      if(0 != WriteBitfileInfoToSubmitFile(fd,
920  1                                           &ebd,
921  1                                           &clm.ce_bitfileID,
922  1                                           clm.ce_mode,
923  1                                           namesize,
924  1                                           (
                                             namesize > 0) ? fname: NULL,
925  1                                           directives_size,
                                                    /* directive_size */
926  1                                           directives_list,
                                                       /* directives */
927  1                                           clm.ce_filesize,
928  1                                           prev_ebd))
929  2      {
930  2        rbe_log_stats(
               0, "** warning: could not submit marked file for restore."
```



```
931  2                           " skipping file %s.", fname);
932  2        return 0;
933  1      }

935  1      memcpy(prev_ebd, &ebd, sizeof(ebfs_uid_ty));

937  1  #if 0     /* G.
                  Sachar: since buf is uninitialized and function is #if 0'd */
938  1      push_to_submitfile(fd, buf, strlen(buf));
939  1  #endif

941  1      return 1;

943  1      /*
944  1       * Creation and destruction of volid's now taks place
945  1       * outside of "go" command.
946  1       *
947  1       * now add bitfile id to volumes needed report
948  1       * if (rcx->ebvlok)
949  1       * {
950  1       *     rcx->ebvllist = ebvl_genvolidlist(
                                              rcx->ebvllist, &clm.ce_bitfileID,
951  1       *                                          1,
                                                ebvl_EbfsIdType_BitFile,
952  1       *                                         &ep_status);
953  1       * }
954  1       * if (rcx->ebvlok && NULL == rcx->ebvllist)
955  1       * {
956  1       *     fprintf(
                   stderr, "Unable to generate volumes needed report, %s (%d)",
957  1       *      e_get_error_text(ep_status), ep_status);
958  1       *      rcx->ebvlok = FALSE;
959  1       * }
960  1       */
961    }         /* end of push_bfinfo_to_submitfile() */
```

```
964     /* Returns: -1 for error encountered
965      *          otherwise number of bytes written
966      *
967      *
968      *
969      */

971     static int
972     push_to_submitfile(int fd,
973                        char *buf,
974                        uint_t nbytes)
975   1 {
976   1 #if 0
977   1     int wrote;
978   1     int save_errno;

980   1     if (debugmode)
981   2     {
982   2          (void)write(fileno(stdout), buf, nbytes);
983   1     }

985   1     wrote = looprw(fd, buf, (int)nbytes, write_no_eintr);
986   1     save_errno = errno;
987   1     if (wrote != (int)nbytes)
988   2     {
989   2       /*
990   2        * short write error
991   2        */
992   2       rbe_log_stats(0, "\n** Trouble writing submitfile.");
993   2       rbe_log_stats(
                    0, "** wanted to write %d, wrote %d", nbytes, wrote);

995   1     }
996   1     errno = save_errno;
997   1     return wrote;
998   1 #endif
999   1     return nbytes;
1000    } /* end of push_to_submitfile() */
```



```
1002    /*
1003     * Convert an EBFS ID to string form.  Elide leading zeros.
1004     */

1006    static void
1007    ebfsid2str_lz(ebfs_uid_ty *ebfsidp,
1008                  register char *buf)
1009  1 {
1010  1     register char *p;
1011  1     register char *q;
1012  1     unsigned long longvals[4];
1013  1     int i;
1014  1     char tmpbuf[ 33 ];
1015  1     char *hexdigits = "0123456789abcdef";


1018  1     memcpy(longvals, ebfsidp, 16);

1020  1     q = tmpbuf;

1022  1     for (i = 0; i < 4; i++)
1023  2     {
1024  2          int j;
1025  2          register unsigned long ul;

1027  2          q += 8;
1028  2          ul = longvals[i];

1030  2          for (j = 0; j < 8; j++)
1031  3          {
1032  3              *--q = hexdigits[ LONG2INT(ul & 0xF) ];
1033  3              ul >>= 4;
1034  2          }
1035  2          q += 8;
1036  1     }

1038  1     tmpbuf[32] = '\0';

1040  1     /*
1041  1      * Skip over leading 0 characters
1042  1      */

1044  1     for (q = tmpbuf; *q == '0'; q++)
1045  2     {
1046  2          /* null */
1047  1     }

1049  1     /*
1050  1      * Copy the rest, up to and including the comma, to output buf
1051  1      */

1053  1     p = buf;
1054  1     while ((*p++ = *q++) != '\0')
1055  2     {
1056  2          /* null */
1057  1     }
1058    } /* end of ebfsid2str_lz() */
```

```
1061    static int
1062    ssID2ebfd(rbssID_t *ssidp,
1063              ebfs_uid_ty *ebfdp)
1064  1 {
1065  1     rbsaveset_t           ss;
1066  1     eperrno               err;


1069  1     /*
1070  1      * clobber it, to make failure to find match obvious
1071  1      */

1073  1    (void)memset((char *)ebfdp, 0, sizeof *ebfdp);


1076  1     if ((err = ss_find1(ssidp, &ss, 0)) == 0)
1077  2     {
1078  2       memcpy(ebfdp, &ss.ss_dirID, sizeof(ebfs_uid_ty));
1079  1     }

1081  1     return err ? -1 : 0;
1082    }       /* end of ssID2ebfd() */
```



```
1086    /*****************************************************************
                                                                        ********
1087    **
1088    ** Routine:  RSTSL_get_catalog_info
1089    **
1090    ** Inputs:  time      - The time of the backup that is being worked
                                                                            with
1091    **
1092    ** Outputs:  level    - reference to the level string to be output
1093    **                     will contain the level of the backup
1094    **           numrec    - reference to the string which will contain the
                                                                            number
1095    **                     of records for the backup
1096    **           catType   - reference to the string which will contain the
                                                                            type
1097    **                     of catalog for the specified backup
1098    **
1099    ** Purpose:  Function to retrieve backup level, catalog type,
                                                                  and number of
1100    **           records and then retun it to the client
1101    **
1102    ** Return Codes: E_SUCCESS   - if the cattalog exists and able to
                                                                        return
1103    **                          information
1104    **               EP_RB_RECOVER_NO_CATALOG  - error getting the catalog
                                                                          info
1105    **
1106    *****************************************************************
                                                                        *******
1107    */
1108    eerrno_ty
1109    RSTSL_get_catalog_info( const time_t time,
1110                            char **level,
1111                            char **numrec,
1112                            char **catType)
1113         /* Called from re_get_catalog_info_1_svc RPC call*/
1114  1 {
1115  1      cat_descriptor *catdPtr;
                                      /* pointer to the catalog descriptor */
1116  1      rbcat_head_t cathdr;
                                /* pointer to head of a catalog retrieved from
1117  1                                * the catalog descriptor
1118  1                                */
1119  1      int plane_count=0;
                                /* used to keep track of the traversal through
1120  1                                * the catalog plane structure
1121  1                                */
1122  1      int number_of_planes;
                                    /* total number of planes to traverse */


1125  1      number_of_planes=-(rcp->rc_nplanes);
                                             /* must be negated because of
1126  1                                           * previous
                                                      implementation of
1127  1                                          * cat struct
1128  1                                          */

1130  1      do  /* traverse the list of planes in the catalog structure */
1131  2         {
1132  2           /* get a pointer to the plane I want staring at 0
1133  2            * and counting down to the number of planes in catalog
                                                                        struct
```

```
1134 2              */
1135 2           catdPtr = mcat_getcatd(rcp->rc_mcp, plane_count);
                                                        /* get the head */
1136 2           if (NULL == catdPtr)  /* if there are no catalogs */
1137 3             {
1138 3               return(EP_RB_RECOVER_NO_CATALOG);
1139 2             }

1141 2           if (0 != catd_get_cat_head(
                              catdPtr, &cathdr))  /* if there is no head
1142 2                                                 * there is
                                                        still not
1143 2                                                 *catalog
1144 2                                                 */
1145 3             {
1146 3               return(EP_RB_RECOVER_NO_CATALOG);
1147 2             }

1149 2           plane_count--;
                          /* decrement counter, must go into negatives */

1151 2           /* while the backup we are looking for has not been found,
                                                                    and
1152 2            * we have not traversed through the entire list
1153 2            */
1154 1           }while ((time !=cathdr.ch_time)&&(
                                       plane_count>=number_of_planes));

1156 1         *numrec=(char *)malloc( ULONG_TO_CHAR_SIZE );
                                                   /* must malloc memory
1157 1                                                * because cannot
                                                          strdup
1158 1                                                * a ulong
1159 1                                                */
1160 1         sprintf(*numrec,"%u",cathdr.ch_nelem);
                                          /* make the ulong a string
1161 1                                                * which is copied
                                                         from the
1162 1                                                *head of the
                                                          catalog
1163 1                                                */
1164 1         (*level) = (char *)calloc(1,2);
1165 1         (*level)[0] = cathdr.ch_level;

1167 1         switch (cathdr.ch_state)
1168 2            {
1169 2            case SSCAT_PARTIAL:
1170 2                 *catType=esl_strdup("PARTIAL");
1171 2                 break;
1172 2            case SSCAT_UNSORTED:
1173 2                 *catType=esl_strdup("UNSORTED");
1174 2                 break;
1175 2            case SSCAT_SORTED:
1176 2                 *catType=esl_strdup("SORTED");
1177 2                 break;
1178 2            case SSCAT_FULL:
1179 2                 *catType=esl_strdup("FULL");
1180 2                 break;
1181 2            case SSCAT_DELTA:
1182 2                 *catType=esl_strdup("DELTA");
1183 2                 break;
1184 2            case SSCAT_EXPIRED:
1185 2                 *catType=esl_strdup("EXPIRED");
1186 2                 break;
1187 2            case SSCAT_NONE:
1188 2                 *catType=esl_strdup("NONE");
```



```
1189 2                 break;
1190 2            default:
1191 2                 *catType=esl_strdup("UNKNOWN");
1192 2                 break;
1193 1            }

1195 1          return E_SUCCESS;
1196       }
```

```
1   /*
2   ** Copyright 1996,1997 EMC Corporation
3   */

5   /*
6   ** EDMProcessManager.c
7   **
8   ** Mission Statement: This is the entry point for the Process Manager
                                                                    thread.
9   **
10  ** Primary Data Acted On:
11  **
12  ** Compile-Time Options:
13  **
14  **                      USE_SUNRPC - Compile source with sunrpc
                                                              support.  If
15  **                                   not set, assume DCE support.
16  **
17  ** Basic idea here: Module for coding the Process Manager thread.
18  */

20  /*
21  ** The following provides an RCS id in the binary that can be located
22  ** with the what(1) utility.  The intent is to keep this short.
23  */
24  #if !defined(lint)
25  static char     RCS_id [] = "@(#)$RCSfile: EDMProcessManager.c,v $ "
26                              "$Revision: 1.23 $ "
27                              "$Date: 1997/02/06 20:49:15 $" ;
28  #endif

30  /* #define _POSIX_SOURCE   unable to compile with this define set */
31  /* #define _XOPEN_SOURCE   unable to compile with this define set */

33  #include <esl/c_portable.h>
34  #include <esl/ep_xopen.h>
35  #include <esl/inout.h>

37  #include <syslog.h>
38  #include <unistd.h>
39  #include <stdlib.h>

41  #include <pthread.h>
42  #include <EDMProcessManager.h>
43  #include <EDMRECommandApi.h>
44  #include <EDMRestoreEngLog.h>
45  #include <EDMmain.h>
46  #include <restore/restore_engine.h>
47  #include <restore/restore_api.h>
48  #include <restore/REprogmsg.h>
49  #include <restore/EDMREProgressApi.h>
50  #include <EDMFinalStatus.h>


53  /* local prototypes */

55  static void unregister_rpc( void );
56  static void start_completion( EDMREGlobalStatus );


59  /* local data */

61  static boolean_ty        completion_signalled = FALSE;

63  struct timeout_array
64 1 {
```



```
65 1    time_t        timeoutlen;
66 1    time_t        guideadlen;
67 1  } tval[MAX_GLOBAL_STATUS_VALUE+1] = {
68 1      // Timeout value        Gui is dead value
69 1    { 5*SECONDS_PER_MINUTE, 5*SECONDS_PER_MINUTE }, // Exiting
70 1    { 5*SECONDS_PER_MINUTE, 5*SECONDS_PER_MINUTE }, // Starting
71 1    { SECONDS_PER_YEAR,     2*SECONDS_PER_HOUR },   // Browsing
72 1    { 3*SECONDS_PER_HOUR,   5*SECONDS_PER_MINUTE }, // Pre phase
73 1    { 10*SECONDS_PER_DAY,   5*SECONDS_PER_MINUTE }, // Execute
74 1    { 2*SECONDS_PER_DAY,    5*SECONDS_PER_MINUTE }  // Post Phase
75      };

77    boolean_ty
78    IsRestoreTimedOut(IN time_t lastguitime, IN time_t incurrentstate,
79                      IN int status)
80 1  {
81 1      time_t t = time(NULL);

83 1      if (status > MAX_GLOBAL_STATUS_VALUE)
84 2      {
85 2          return FALSE;
86 1      }

88 1      if (status < 0)
89 2      {
90 2          status = 0;       /* all exiting conditions use same timeout */
91 1      }

93 1      if ( (t - tval[status].guideadlen) > lastguitime)
94 2      {
95 2          return TRUE;
96 1      }
97 1      else if ( (t - tval[status].timeoutlen) > incurrentstate)
98 2      {
99 2          return TRUE;
100 1     }

102 1     return FALSE;
103   }
```

```
105     void *
106     REProcessManager(void *buff)
107   1 {
108   1     int  status;
109   1     int  command;
110   1     int  result;
111   1     void         *input_ptr;
112   1     void         *output_ptr;
113   1     boolean_ty   finish_rpc_recvd = FALSE;
114   1     boolean_ty   reader_finish_rcvd = FALSE;
115   1     EDMREGlobalStatus    internal_status;
116   1     time_t       status_time;

118   1     setGlobalStatus( EDMRE_STATE_STARTING );

120   1     while (
      !reader_finish_rcvd || !finish_rpc_recvd )   /* until time to exit */
121   2     {
122   2         /* wait for next command */
123   2         if (PopCommand( 1, &command, &status ))
124   3         {
125   3             if (COMMAND_RECORD_GET_FAILED != status)
126   3             /* log error if not 'normal' queue empty error */
127   3                 EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
128   3                                       MESSAGE_RECORD_GET_FAILED, 0,
129   3                                       "PopCommand failed;
                                                              status = %d",
130   3                                       status );

132   3             /* check for completion timeout or idle timeout */
133   3             internal_status = getGlobalStatus( &status_time );
134   3             if (completion_signalled)
135   4             {
136   4                 if (TRUE == IsRestoreTimedOut(  getLastRpcTime(
                                                              ), status_time,
137   4                                               internal_status ) )
138   5                 {
139   5                     if (!finish_rpc_recvd)
140   6                     {
                          /* let restore service module clean up, stop rpcs */
141   6                         result = EDMRE_Finish( NULL, NULL );
142   6                         unregister_rpc( );      /* cleanup csc i/f */
143   5                     }
144   5                     EDMRestoreEng_logent(
                                              __FILE__, __LINE__, LOG_ERR,
145   5                                 MESSAGE_SHUTDOWN, 0,
146   5                                 "Shutting down after timeout awaiting
                                                              sync" );
147   5                     break;
                                                  /* escape while to exit */
148   4                 }
149   3             } else
150   4             {
151   4                 if (TRUE == IsRestoreTimedOut( getLastRpcTime(
                                                              ), status_time,
152   4                                               internal_status ) )
153   5                 {   /* if already exiting, leave state alone */
154   5                     if (internal_status > 0)
155   5                         start_completion( EDMRE_STATE_TIMEOUT );
156   5                     else
157   5                         start_completion( internal_status );
158   4                 }
159   3             }
160   3             continue;
                    /* keep waiting in case thread wait got interrupted */
```



```
161   2         }

163   2         /* got some command; see if we're in completion sequence: */
164   2         if (completion_signalled)
165   3         {
166   3             if (COMMAND_FINISH == command && !finish_rpc_recvd)
167   4             {
168   4                 if (PopRpcInput( &input_ptr, &status ))
169   5                 {
170   5                     EDMRestoreEng_logent(
                                              __FILE__, __LINE__, LOG_ERR,
171   5                                   MESSAGE_POP_RPC_INPUT_FAILED, 0,
172   5                                   "PopRpcInput failed;
                                                      status = %d", status );
173   4                 }
174   4                 else
175   4                     /* let restore service module clean up;
                                                          stop rpc's */
176   4                     result = EDMRE_Finish( input_ptr, &output_ptr );

178   4                 finish_rpc_recvd = TRUE;
179   4                 unregister_rpc( );
180   3             }
181   3             else if (
                  COMMAND_READER_FINISHED == command && !reader_finish_rcvd)
182   4             {
183   4                 reader_finish_rcvd = TRUE;
184   5                 if (!finish_rpc_recvd) {
185   5                 /* let restore service module clean up, stop rpcs */
186   5                     result = EDMRE_Finish( NULL, NULL );
187   5                     unregister_rpc( );  /* cleanup csc i/f */
188   4                 }
189   4                 break;          /* exit */
190   3             }
191   3             else
192   4             {
193   4                 EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
194   4                                       MESSAGE_INVALID_COMMAND, 0,
                                                          "cmd value: %d",
195   4                                       command );
196   3             }
197   3             continue;
                    /* check if both finishes recvd, else keep waiting */
198   2         }

200   2         /* not in completion seq;
                              get pointer to rpc input argument structure */
201   2         if (PopRpcInput( &input_ptr, &status ))
202   3         {
203   3             EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
204   3                                   MESSAGE_POP_RPC_INPUT_FAILED, 0,
205   3                               "PopRpcInput failed;
                                                  status = %d Lost command: %d",
206   3                                   status, command );
207   3             continue;   /* ??? keep trying or return ?? */
208   2         }

210   2         switch( command )
211   3         {
212   3         case COMMAND_GET_RESTORABLE_OBJECTS:
213   3             result = EDMRE_GetRestorableObjects(
                                                  input_ptr, &output_ptr );
214   3             break;
215   3         case COMMAND_MARK_OBJECT:
216   3             result = EDMRE_MarkObject( input_ptr, &output_ptr );
217   3             break;
```

```
218 3           case COMMAND_UNMARK_OBJECT:
219 3               result = EDMRE_UnmarkObject( input_ptr, &output_ptr );
220 3               break;
221 3           case COMMAND_SUBMIT:
222 3               result = EDMRE_Submit( input_ptr, &output_ptr );
223 3               break;

225 3           case COMMAND_START:
226 3               result = EDMRE_Start( input_ptr, &output_ptr );
227 3 #if 0       /* taken out to allow continuation after successful & aborted
                                                                    restore */
228 3               start_completion( getGlobalStatus(NULL) );
                                                    /* leave same state */
229 3 #endif
230 3               break;

232 3           case COMMAND_FIND_RESTORABLE_OBJECTS:
233 3               result = EDMRE_FindRestorableObjects(
                                                input_ptr, &output_ptr );
234 3               break;

236 3           case COMMAND_FINISH:
237 3               result = EDMRE_Finish( input_ptr, &output_ptr );
238 3               finish_rpc_recvd = TRUE;
239 3               if (EDMRE_STATE_SUCCESSFUL
240 3                   < (internal_status = getGlobalStatus(
                                                        &status_time ) )
241 3               )
242 3                   /* if already exiting, leave state alone */
243 3                   start_completion( EDMRE_STATE_SUCCESSFUL );
244 3               else
245 3                   start_completion( internal_status );
246 3               unregister_rpc( );
247 3               break;              /* await dispatcher finish command */

249 3           case COMMAND_LOAD_RECX_DIRECTIVES:
250 3               result = EDMRE_Load_recx_directives(
                                                input_ptr, &output_ptr );
251 3               break;
252 3           case COMMAND_GET_ALL_TIMES:
253 3               result = EDMRE_GetAllBackupTimes(
                                                input_ptr, &output_ptr );
254 3               break;
255 3           case COMMAND_SET_PREVIOUS_BACKUP:
256 3               result = EDMRE_SetPreviousBackup(
                                                input_ptr, &output_ptr );
257 3               break;
258 3           case COMMAND_SET_NEXT_BACKUP:
259 3               result = EDMRE_SetNextBackup( input_ptr, &output_ptr );
260 3               break;
261 3           case COMMAND_SET_FIRST_BACKUP:
262 3               result = EDMRE_SetFirstBackup( input_ptr, &output_ptr );
263 3               break;
264 3           case COMMAND_SET_MOST_RECENT_BACKUP:
265 3               result = EDMRE_SetMostRecentBackup(
                                                input_ptr, &output_ptr );
266 3               break;
267 3           case COMMAND_SET_BACKUP_FOR_TIME:
268 3               result = EDMRE_SetBackupForTime( input_ptr, &output_ptr );
269 3               break;
270 3           default:
271 3               EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
272 3                                     MESSAGE_INVALID_COMMAND, 0,
                                                        "cmd value: %d",
273 3                                     command );
274 3               result = COMMAND_RESULT_FAILURE;
```



```
275 2           }

277 2           /* push result arg structure pointer, IF command succeeded */
278 2           if( result != COMMAND_RESULT_FAILURE)
279 3           {
280 3               if (PushRpcOutput( output_ptr, &status) )
281 4               {
282 4                   EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
283 4                                         MESSAGE_PUSH_RPC_OUTPUT_FAILED,
                                                                    0,
284 4                                         "PushRpcOutput failed;
                                                        status = %d",
285 4                                         status );
286 3               }
287 2           }

289 2           if (PushResult( result, command, &status) )
290 3           {
291 3               EDMRestoreEng_logent( __FILE__, __LINE__, LOG_ERR,
292 3                                     MESSAGE_FAILURE_TO_QUEUE_RESULT, 0,

293 3                                     "PushResult failed;
                                                status = %d", status );
294 2           }
295 1       }

297 1   #if 0    /* I think we just leave global status as its already set */
298 1       if ( reader_finish_rcvd && finish_rpc_recvd )        /*  good exit
                                                                    */
299 1           setGlobalStatus( /* good exit ?? */ 0 );
300 1   #endif

302 1       exit ( getGlobalStatus(NULL) );
303 1       return buff;
304     }
```

```
307   /* local function to unregister rpc interface */

309   static void unregister_rpc( void )
310 1 {
311 1           sleep( 1 );
                          /* allow last rpc (finish) response to get sent */
312 1           unregister_csc( );      /* stop RPC traffic */
313 1           return;
314   }
```



```
317   /* local function to start completion sequence */

319   static void start_completion( EDMREGlobalStatus status )
320 1 {
321 1           setGlobalStatus( status );
322 1           SendFinalStatus( );              /* signal dispatcher */
323 1           completion_signalled = TRUE;
324 1           return;
325   }
```

```
/*
** Copyright 1996,1997 EMC Corporation
*/

/*
** EDMRestoreEng.c
**
** Mission Statement: This is the main service file for the EDMsessd
                                                        daemon. This
**                    file contains the callbacks from the main
                                                      function which
**                    prepares the daemon to go off and service RPC's.
**
** Primary Data Acted On:
**
** Compile-Time Options:
**
**                      USE_SUNRPC - Compile source with sunrpc
                                                      support.  If
**                                   not set, assume DCE support.
**
** Basic idea here: Module for UNIX specific daemon initialization
*/

/*
** The following provides an RCS id in the binary that can be located
** with the what(1) utility.  The intent is to keep this short.
*/
#if !defined(lint)
static char     RCS_id [] = "@(#)$RCSfile: EDMsessdd.c,v $ "
                            "$Revision: 1.23 $ "
                            "$Date: 1997/02/06 20:49:15 $" ;
#endif

/* #define _POSIX_SOURCE   unable to compile with this define set */
/* #define _XOPEN_SOURCE   unable to compile with this define set */

#include <esl/c_portable.h>
#include <esl/ep_xopen.h>
#include <esl/inout.h>

#include <stdarg.h>
#include <string.h>
#include <syslog.h>
#include <pthread.h>
#include <thread.h>
#include <sys/utsname.h>
#include <netdb.h>

#include <logging/logging.h>
#include <util/esl_core.h>
#include <util/esl_pidfile.h>
#include <util/esl_daemon.h>
#include <csc/cscomm.h>

#include <restore/csc_EDMRestoreEng.h>

#include <EDMmain.h>
#include <EDMRestoreEngLog.h>
#include <EDMProcessManager.h>
#include <EDMProgress.h>
#include <EDMRE_ccr.h>
#include <EDMRE_ccw.h>
#include <EDMRECommandApi.h>
#include <EDMREQuestionApi.h>
#include <EDMREDrainApi.h>
```



```
/*
 * Need to define _XOPEN_SOURCE for signal funtion definitions
 * and certain signal structure definitions.
 */
#define _XOPEN_SOURCE

#include <signal.h>

#undef _XOPEN_SOURCE

static rpc_if_handle_t if_spec;

static int       G_debug = FALSE;
                                /* Variable which will disable forking */

static char      **commandlineargs; /* Pointer to command line args */

/*****************************************************************************
                                                                    ********
**
** Routine: IsDebugOn
**
** Inputs: None
**
** Outputs:  None
**
** Return Codes:
**               TRUE if debug is on.
**
** Purpose:      This routine can be used to tell other subsystems
**               whether debugging is available.
**
** Intended caller:  internal only.
**
*****************************************************************************
                                                                    ********
*/
boolean_ty
IsDebugOn()
{
#ifdef DEBUG
    return TRUE;
                          /* if DEBUG defined, we must be in debug mode */
#endif
    if (debugmode)        /* if turned on manually via adb, its on */
        return TRUE;

    return G_debug;
                    /* default is how we were started: -d menas debug */
}
```

```
111  /*****************************************************************************
                                                                          ********
112  **
113  ** Routine:kill_handler
114  **
115  ** Inputs: int sigval - the signal which was receieved.
116  **
117  ** Outputs:  Will log messages telling what action is being taken.
118  **
119  ** Return Codes:
120  **               exits with the number of the signal received
121  **
122  ** Purpose:      This routine handles specific signals i.e. SIGINT,
123  **               SIGQUIT,
                          SIGTERM. Each results in a log entry and an exit.
124  **
125  ** Intended caller:  internal only.
126  **
127  *****************************************************************************
                                                                          ********
128  */
129  static void kill_handler( IN int sigval )
130 1 {
131 1 error_status_t   status;
132 1 time_t           current_time;
133 1 char             *ctimebuf;
134 1 char             *ebuff = NULL;

136 1   /* If main exits, it calls this routine with signal 0 */

138 1   /* Unregister the interface */
139 1   (void) csc_unregister_server_interface(&if_spec, &status);

141 1   /* If the unregister fails, report the problem, but continue */
142 1   if ( status != error_status_ok )
143 2   {
144 2     ebuff = (char *) csc_get_error( status );

146 2     (void) EDMRestoreEng_logent(
                       __FILE__,__LINE__, LOG_ERR, MESSAGE_NO_LOGIN, 0,
147 2                    "CSC_SERVER_LOGIN failed: <%d> %s",
148 2                    status, (ebuff ? ebuff : "Unknown error") );
149 1   }

151 1   /* Get the current time */
152 1   (void) time(&current_time);

154 1   ctimebuf = ctime(&current_time);

156 1   /* Overlay newline with null - buf should always be 26 bytes long */
157 1   ctimebuf[ strlen(ctimebuf) - 1 ] = 0;

159 1   (void) EDMRestoreEng_logent(
                      __FILE__,__LINE__, LOG_INFO, MESSAGE_SHUTDOWN, 0,
160 1                 "Shutting down at %s due to signal %d", ctimebuf,
                                                              sigval);

162 1   exit(sigval);

164  } /* End of kill_handler() */
```



```
167  /*****************************************************************************
                                                                          ********
168  **
169  ** Routine: unregister_csc
170  **
171  ** Inputs: none
172  **
173  ** Outputs:  Will log messages telling what action is being taken.
174  **
175  ** Return Codes:
176  **               none
177  **
178  ** Purpose:      This routine handles the csc_unregister call
179  **
180  ** Intended caller:  internal and process manager before exit
181  **
182  *****************************************************************************
                                                                          ********
183  */
184  void unregister_csc( void )
185 1 {
186 1 error_status_t   status;
187 1 char     *ebuff = NULL;

189 1   /* Unregister the interface */
190 1   (void) csc_unregister_server_interface(&if_spec, &status);

192 1   /* If the unregister fails, report the problem, but continue */
193 1   if ( status != error_status_ok )
194 2   {
195 2     ebuff = (char *) csc_get_error( status );

197 2     (void) EDMRestoreEng_logent(          __FILE__,__LINE__, LOG_ERR,
198 2                                 MESSAGE_CANNOT_UNREGISTER, 0,
199 2                                 "CSC_UNREGISTER_SERVER failed: <%d> %s",
200 2                                 status, (ebuff ? ebuff : "Unknown error") );
201 1   }

203 1   return;
204  }
```

```
206  /*****************************************************************************
                                                                          ********
207   * Function Name:
208   *       display_usage
209   *
210   * Simply displays the usage
211   *
212   * Call Arguments:
213   *       Program name
214   *
215   * Error Outputs and Side Effects:
216   *       Prints usage.
217   *
218   * Special Considerations:
219   *       None.
220   *
221   *****************************************************************************
                                                                          ********
222   */
223  static void
224  display_usage (IN char *progname)
225 1 {
226 1   /* Print out usage stmt.  */

228 1   fprintf (stderr,"Usage: %s [-d]\n", progname);
229 1   fprintf (
       stderr,"-d keep the daemon from forking so debugging is easier\n");

231  } /* end display_usage () */
```



```
234  /*****************************************************************************
                                                                          ********
235   **
236   ** Routine: daemon_catch_interrupts
237   **
238   ** Inputs:      None
239   **
240   ** Outputs:     None
241   **
242   ** Return Codes:
243   **              None
244   **
245   ** Purpose:     Sets up signals for service. On NT we will have to
246   **              consider what OS constructs to replace signals with.
247   **              In this case we are catching SIGTERM, SIGINT, and
248   **              SIGQUIT and ignoring anything else.
249   **
250   ** Intended caller:  internal only.
251   **
252   *****************************************************************************
                                                                          ********
253   */
254  void daemon_catch_interrupts()
255 1 {
256 1 struct sigaction          sactions;       /* Signal actions */

258 1          ZERO( sactions );

260 1         /*
261 1          * Set an empty list so we can set signals we want to handle
262 1          */
263 1          (void) sigemptyset( &sactions.sa_mask );

265 1         /*
266 1          * Add signals that we want to handle
267 1          */
268 1          (void) sigaddset( &sactions.sa_mask, SIGTERM );
269 1          (void) sigaddset( &sactions.sa_mask, SIGINT );
270 1          (void) sigaddset( &sactions.sa_mask, SIGQUIT );

272 1          /* Setup the signal handler. */
273 1          sactions.sa_handler = kill_handler;

275 1         /*
276 1          * Assign handler to each signal we are interested in.
277 1          */
278 1          (void) sigaction( SIGTERM, &sactions, NULL );
279 1          (void) sigaction( SIGINT,  &sactions, NULL );
280 1          (void) sigaction( SIGQUIT, &sactions, NULL );

282 1          /*
283 1          * Setup mask so we can specify what signals we will ignore.
284 1          */
285 1          (void) sigfillset( &sactions.sa_mask );

287 1         /*
288 1          * We want to ignore everything except those we have set up
289 1          * above so remove those from the list.
290 1          */
291 1          (void) sigdelset( &sactions.sa_mask, SIGTERM );
292 1          (void) sigdelset( &sactions.sa_mask, SIGINT );
293 1          (void) sigdelset( &sactions.sa_mask, SIGQUIT );

295 1         /*
```

```
296 1          * Set the mask. Since no other threads have been started,
297 1          * all threads will get this mask.
298 1          */
299 1          (void) thr_sigsetmask( SIG_SETMASK, &sactions.sa_mask, NULL );
300    }
```



```
303    /*****************************************************************************
                                                                          ********
304    **
305    ** Routine: daemon_check_proper_ID
306    **
307    ** Inputs:      None
308    **
309    ** Outputs:     None
310    **
311    ** Return Codes:
312    **              exits with an error when the user is not root
313    **
314    ** Purpose:     Checks user's ID and determines if the user is allowed
315    **              to execute service.
                                    If there are no constraints then this
316    **              function may be blank.
317    **
318    ** Intended caller:  internal only.
319    **
320    *****************************************************************************
                                                                          ********
321    */

323    void daemon_check_proper_ID()
324 1  {
325 1          /*
326 1          ** Check for root
327 1          */

329 1          if (geteuid() != E_ROOTUID)
330 2          {
331 2            (void) EDMRestoreEng_logent(
                        __FILE__,__LINE__, LOG_ERR, DAEMON_NOTSUPERUSER, 0,
332 2                          "Must be run as superuser, uid was %d",
333 2                          geteuid());
334 2            exit(1);
335 1          }
336    }
```

```
338  /*****************************************************************************
                                                                          ********
339  **
340  ** Routine: parse_commandline
341  **
342  ** Inputs:      argc, argv (command line arguments)
343  **
344  ** Outputs:     None
345  **
346  ** Return Codes:
347  **              exits with an error when the user types a bad argument
348  **
349  ** Purpose:     Parses command line arguments and sets flags. If there
350  **              are no flags to be set then this function may be empty.
351  **
352  ** Intended caller:  internal only.
353  **
354  *****************************************************************************
                                                                          ********
355  */

357  void parse_commandline(int argc, char *argv[])
358 1 {
359 1 int                         opt;            /* Process options   */

361 1          commandlineargs = argv;

363 1          while ((opt = getopt(argc,argv,"dD")) != EOF )
364 2          {
365 2            switch(opt)
366 3            {
367 3              case 'd':
368 3              case 'D':
369 3                  G_debug = TRUE;
370 3                  debugmode = 1;
                                        /* turn on other debugmode flag */
371 3                  break;
372 3              default:
373 3                  (void) display_usage( argv[0] );
374 3                  exit(1);

376 2            }
377 1          }
378  }
```



```
380  /*****************************************************************************
                                                                          ********
381  **
382  ** Routine: daemon_initialize_logging
383  **
384  ** Inputs:      None
385  **
386  ** Outputs:     None
387  **
388  ** Return Codes:
389  **              None
390  **
391  ** Purpose:     Do whatever it takes to initialize logging.
                                                                    In the near
392  **              future this may involve doing something with catalogs
                                                                           or
393  **              calling higher level logging functions which
                                                                  encapsulate
394  **              these things.
395  **
396  ** Intended caller:  internal only.
397  **
398  *****************************************************************************
                                                                          ********
399  */

401  void
402  daemon_initialize_logging()
403 1 {
404 1          /* Pass in argv[0], the program name */
405 1          (void) esl_log_init(commandlineargs[0]);
406  }
```

```
408  /*************************************************************************
                                                                      ********
409  **
410  ** Routine: daemon_become_daemon
411  **
412  ** Inputs:       None
413  **
414  ** Outputs:      None
415  **
416  ** Return Codes:
417  **               exits with an error code if initialization fails
418  **
419  ** Purpose:      This function is for doing the forking etc. under UNIX.
420  **               It is unknown what will be necessary under NT.
421  **
422  ** Intended caller:  internal only.
423  **
424  *************************************************************************
                                                                      ********
425  */

427  void
428  daemon_become_daemon()
429 1 {
430 1          char *ptr;
431 1         int  ret = 0;

433 1         /*
434 1          * Strip the path from the program name so we can use it
435 1          * elsewhere.
436 1          */
437 1         ptr = strrchr(commandlineargs[0], '/');
438 1         if (ptr == NULL)
439 1             ptr = commandlineargs[0];
440 1         else
441 1             ptr++;

443 1         /* Change directory to a process specific core directory */
444 1         ret = esl_coredir_setup(ptr);
445 1         if (ret != 0)
446 2         {
447 2              (void) EDMRestoreEng_logent( __FILE__,__LINE__,LOG_ERR,
448 2                                  MESSAGE_ERR_IN_ESL_COREDIR, errno,
449 2                                  "esl_coredir_setup failed" );
450 2             exit(1);
451 1         }

453 1          /*
454 1          **  This is now esl functionality.
                                        This code does everything necessary
455 1          **  to make this a "real" daemon by detaching from the
                                                              terminal
456 1          **  changing the process group, closing stdout, stderr, stdin,
                                                                        ...
457 1          */

459 1 /***     if (G_debug == FALSE)
460 1         {
461 1             ret = esl_daemon_startup();
462 1              if (ret != 0)
463 1             {
464 1                 fprintf(
                          stderr, "%s: Failed to initialize as daemon.\n",
465 1                       commandlineargs[0]);
466 1                 exit(1);
```



```
467 1             }
468 1         }    **/
469  }
```

```
471  /*****************************************************************************
                                                                          ********
472  **
473  ** Routine: rpc_init
474  **
475  ** Inputs:      None
476  **
477  ** Outputs:     None
478  **
479  ** Return Codes:
480  **              exits with an error code if initialization fails
481  **
482  ** Purpose:     This function is for doing RPC initialization.
483  **              For the most part it involves calling the csc routines.
484  **              This is pretty standard between UNIX and NT.
485  **
486  ** Intended caller:  internal only.
487  **
488  *****************************************************************************
                                                                          ********
489  */

491  void rpc_init()
492 1 {
493 1 error_status_t           status;
                                        /* error status (nbase.h) */
494 1 unsigned char    *conn_h;
495 1 struct hostent  *hp;
496 1 char   *ebuff;
497 1 struct utsname name;

499 1 /*
500 1 ** This is here because of HP which may or may not define timeval.
501 1 ** May be removed when esl_timeval is ported to clients
502 1 */
503 1 #ifdef _STRUCT_TIMEVAL
504 1          struct timeval  sleep_interval = {5,0};
                                        /* 5 second sleep interval */
505 1 #else
506 1          struct timespec sleep_interval = {5,0};
                                        /* 5 second sleep interval */
507 1 #endif

509 1          /* Setup the interface specification for RPC */

511 1          RE_SERVER_IFSPEC(if_spec);

513 1          /*
514 1          * Login as SERVER_PRINCIPAL.  The context of the process
515 1          * will be set to this principal.
516 1          *
517 1          * This process will keep trying to login to DCE if the
                                                                    registery
518 1          * server is unavailable.
                                Note that under SUN RPC this is a no-op.
519 1          */
520 1          while (TRUE)
521 2          {
522 2                  (void) csc_server_login(RE_SERVER_PRINCIPAL,
523 2                                          RE_SERVER_KEYTAB, &status);

525 2                  /* If we succeeded, then exit this loop.  */
526 2                  if ( status == error_status_ok )
527 3                  {
528 3                          break;
```



```
529 2                  }
530 2                  else    /* Print error message if appropriate. */
531 3                  {
532 3                   ebuff = (char *) csc_get_error( status );

534 3                     (void) EDMRestoreEng_logent(
                                                __FILE__,__LINE__,LOG_ERR,
535 3                                        MESSAGE_NO_LOGIN, 0,
536 3                                        "CSC_SERVER_LOGIN failed: <%d>
                                                                        %s",
537 3                                        status, (
                                          ebuff ? ebuff : "Unknown error"));
538 2                  }

540 2                  /* If the failure was due to unavailable client,
541 2                  * pause and then try again.
542 2                  */
543 2                  if (status == sec_rgy_server_unavailable)
544 3                  {
545 3                          /*
546 3                           * uses sleep when SUNRPC, otherwise uses
547 3                           * pthread call to delay for the specified
548 3                           * time
549 3                           */
550 3                             CSC_SLEEP(sleep_interval);
551 3                             continue;
552 2                  }

554 2                  /* If we got here, we had a unexpected failure.  */
555 2                  (void) EDMRestoreEng_logent(
                                                __FILE__,__LINE__,LOG_ERR,
556 2                                       MESSAGE_NO_LOGIN, 0,
557 2                                       "The service cannot log in as
                                                                required");


559 2                  exit(1);
560 1          }

562 1          uname(&name);

564 1          hp = gethostbyname(name.nodename);

566 1          if (hp == NULL)
567 2          {
568 2               (void) EDMRestoreEng_logent( __FILE__,__LINE__,LOG_ERR,
569 2                                            MESSAGE_GETHOSTBYNAME_FAIL,
                                                                    errno,
570 2                                            "gethostbyname failed" );
571 2               exit(1);
572 1          }

574 1          memcpy((char *) &if_spec.ip_addr, hp -> h_addr, hp->h_length);

576 1           /*
577 1           ** We need to initialize the authorization module before we
                                                                        do
578 1           ** a listen.
579 1           */
580 1           (void)csc_authorization_init(&status);

582 1           if ( status != error_status_ok )
583 2           {
584 2            ebuff = (char *) csc_get_error( status );

586 2              (void) EDMRestoreEng_logent( __FILE__,__LINE__,LOG_ERR,
587 2                                           MESSAGE_NOAUTHORIZATION, 0,
```

```
588 2                                "CSC_AUTHORIZATION_INIT failed: <%d> %s",
589 2                                 status, (
                                           ebuff ? ebuff : "Unknown error") );
590 2             exit(1);
591 1           }

593 1          conn_h = calloc(1, CONNECT_HANDLE_SIZE);

595 1          if (conn_h == NULL)
596 2          {
597 2             (void) EDMRestoreEng_logent( __FILE__,__LINE__,LOG_ERR,
598 2                                 MESSAGE_NO_MEMORY, 0,
599 2                                 "Failure allocating memory for connection
                                                                    handle");
600 2             exit(1);
601 1          }

603 1           (void) csc_register_private_server_interface(
                                                    &if_spec,
604 1                                               0,
605 1                                               1,
606 1                                               conn_h,
607 1                                               &status);

609 1           if ( status != error_status_ok )
610 2           {
611 2            ebuff = (char *) csc_get_error( status );

613 2             (void) EDMRestoreEng_logent( __FILE__,__LINE__,LOG_ERR,
614 2                                 MESSAGE_CANNOTREGISTER, 0,
615 2                                "CSC_REGISTER_SERVER_INTERFACE failed:
                                                                    <%d> %s",
616 2                                 status, (
                                           ebuff ? ebuff : "Unknown error") );
617 2             exit(1);
618 1           }

620 1          free(conn_h);
621        }
```



```
623    /*************************************************************************
                                                                        ********
624    **
625    ** Routine: rpc_run
626    **
627    ** Inputs:       None
628    **
629    ** Outputs:      None
630    **
631    ** Return Codes:
632    **               None
633    **
634    ** Purpose:      This function is for running the RPC listen.
635    **               This is pretty standard between UNIX and NT.
636    **
637    ** Intended caller:  internal only.
638    **
639    *************************************************************************
                                                                        ********
640    */

642    void rpc_run()
643 1  {
644 1  error_status_t            status;
                                          /* error status (nbase.h) */
645 1  char  *ebuff;

647 1           /* listen for RPC calls forever. */
648 1           (void) csc_server_listen(
                                rpc_c_listen_max_calls_default, &status );

650 1          ebuff = (char *) csc_get_error( status );

652 1           /* We don't expect to get here. */
653 1           (void) EDMRestoreEng_logent( __FILE__,__LINE__,LOG_ERR,
654 1                                MESSAGE_SERVERLISTEN, 0,
655 1                                "CSC_SERVER_LISTEN failed: <%d> %s",
656 1                                 status, (
                                           ebuff ? ebuff : "Unknown error") );

657    }
```

```
659  /*****************************************************************
                                                                ********
660  **
661  ** Routine: daemon_specific_initialization
662  **
663  ** Inputs:       None
664  **
665  ** Outputs:      None
666  **
667  ** Return Codes:
668  **               None
669  **
670  ** Purpose:      Do whatever makes this daemon special.
                                                        In some cases you
671  **               may want to start a thread or open a socket.
                                                          Do that here.
672  **
673  ** Intended caller:  internal only.
674  **
675  *****************************************************************
                                                                ********
676  */

678  void
679  daemon_specific_initialization()
680 1 {
681 1 int                     status;
                                        /* error status (nbase.h) */
682 1 void                    *statptr;
                                        /* error status (nbase.h) */
683 1 int                     ret;
684 1 pthread_t               pmtid;
685 1 pthread_t               progresstid;
686 1 pthread_t               ccrtid;
687 1 pthread_t               ccwtid;
688 1 time_t   current_time;
689 1 char     *ctimebuf;

691 1           ret = CommandAPIInit(&status);
692 1           ret = QuestionAPIInit(&status);
693 1           ret = DrainAPIInit(&status);

695 1           /* Find out what time it is */
696 1           (void) time(&current_time);

698 1           ctimebuf = ctime(&current_time);

700 1           /* Overlay newline with null - buf should always be 26 bytes
                                                              long */
701 1           ctimebuf[ strlen(ctimebuf) - 1 ] = 0;

703 1           /* Log startup message */
704 1           (void) EDMRestoreEng_logent( __FILE__, __LINE__, LOG_INFO,
705 1                           MESSAGE_STARTUP, 0,
                                   "Restore service %s starting up at %s",
706 1                           commandlineargs[0], ctimebuf );

708 1          /*
709 1           * Start the other threads in the daemon. The main thread
710 1           * becomes the RPC thread. REProcessManager is the
711 1           * entry point for the periodic event thread.
712 1           */
713 1          pthread_create(&pmtid, NULL, REProcessManager, NULL);
714 1          pthread_create(&progresstid, NULL, REProgress, NULL);
715 1          /* pthread_create(&ccrtid, NULL, RestoreSvc_ccr, NULL); */
```



```
716 1           pthread_create(&ccwtid, NULL, RestoreSvc_ccw, NULL);


719 1          rpc_init();
720 1           RestoreSvc_Setup();
721 1          rpc_run();

723 1          pthread_join(pmtid, &statptr);
724    }
```

```
726  /*****************************************************************
                                                                ********
727  **
728  ** Routine: daemon_cleanup
729  **
730  ** Inputs:       None
731  **
732  ** Outputs:      None
733  **
734  ** Return Codes:
735  **               None
736  **
737  ** Purpose:      Call function which will clean up daemon properly.
738  **
739  ** Intended caller:  internal only.
740  **
741  *****************************************************************
                                                                ********
742  */

744  void
745  daemon_cleanup()
746 1{
747 1        kill_handler( 0 );
748  }
```